UNIVERSIDAD DE LA LAGUNA

Integración de técnicas de análisis y clasificación de datos mediante un sistema basado en el conocimiento para el diagnóstico de patologías cerebrales y dislexias

Autor: Sigut Saavedra, José Francisco

Director: José Demetrio Piñeiro Vera

Departamento de Física Fundamental y Experimental, Electrónica y Sistemas

Para llevar a cabo la realización de un trabajo de la envergadura de una tesis doctoral, hace falta, aparte de un gran esfuerzo, el apoyo de muchas personas sin las cuales sería aún más dificil alcanzar con éxito los objetivos planteados. Me gustaría aprovechar estas líneas para mostrar mi agradecimiento a algunas de estas personas.

En primer lugar, quisiera agradecer al Dr. D. José Demetrio Piñeiro Vera su excelente labor de dirección, su profesionalidad, su rigurosidad y sobre todo su dedicación a este trabajo, sin la cual no hubiera sido posible su conclusión.

Al Dr. D. Lorenzo Moreno Ruiz, por hacer posible mi carrera universitaria y contribuir a mi formación docente e investigadora.

A la Dra. Dña Soledad Mañas Alcón, por proporcionarnos los datos de pacientes con Alzheimer, que han servido para validar el trabajo realizado.

Al Dr. D. Juan E. Jiménez González, por proporcionarnos los datos de niños con dislexia que también han sido utilizados para la validación del trabajo.

A los miembros del grupo de Computadoras y Control: Dña. Marta Sigut Saavedra, Dr. D. Juan Albino Méndez , Dña. Silvia Alayón Miranda, Dra. Dña. Carina Soledad González, Dr. D. Alberto Hamilton Castro, Dr. D. Graciliano Nicolás Marichal Plasencia, Dr. D. Jose Ignacio Estévez Damas, Dra. Dña. Rosa María Aguilar Chinea, D. Juan Julián Merino Rubio, D. Roberto Marichal Plasencia, D. Evelio José González González, D. Hector Reboso Morales, D. Santiago Torres Álvarez, Dr. D. Jose Luis Sánchez de la Rosa, Dr. D. Leopoldo Acosta Sánchez, D. Carlos Martín Galán, D. Roberto Betancor Bonilla, D. Sergio Hernández Alonso, D. Agustín José Padrón, D. Manuel Fernández Vera.

A Helen por apoyarme en estos últimos y duros meses de trabajo.

Especialmente, a mis padres y a mi hermana que lo son todo para mí y a los que tengo que agradecer todo lo que soy.

A mis padres y a mi hermana

A Helen

ÍNDICE

INTRODUCCIÓN

La presente memoria supone la continuación de una línea de investigación iniciada desde principios de los 90 por miembros del Grupo de Computadoras y Control del Departamento de Física Fundamental y Experimental, Electrónica y Sistemas. Se trata de una serie de trabajos orientados al diagnóstico de patologías cerebrales a través de la aplicación de diversas técnicas informáticas. Dentro de esta línea, aunque con una perspectiva más general, el objetivo de este trabajo es la automatización del diagnóstico, integrando técnicas de análisis y clasificación de datos mediante un Sistema Basado en el Conocimiento.

Diagnóstico y clasificación son términos que están muy relacionados, ya que la tarea de diagnosticar una determinada enfermedad, es equivalente a clasificar al sujeto como afectado o no por una determinada patología. Desde esta perspectiva, el problema de automatización del diagnóstico se traduce en la automatización del proceso de diseño de un clasificador. En ocasiones, como en los dominios tratados en este trabajo, se puede restringir el problema del diseño del clasificador a utilizar el cuerpo de conocimiento general correspondiente al reconocimiento de patrones y análisis de datos,

derivado a su vez de múltiples disciplinas: estadística, probabilidad, teoría de la decisión, aprendizaje automático, inteligencia artificial, neurociencias, …. Esta restricción es posible debido a que la relación entre los datos y sus categorías no es demasiado compleja o abstracta y, a veces, no existe otra alternativa de análisis debido a la ignorancia sobre el dominio del problema. Aún así, el diseño de un clasificador (en este sentido más restringido) es una tarea compleja, cuya resolución implica, la aplicación de heurísticas y estrategias dependientes de los datos particulares. Por este motivo, se ha considerado conveniente su integración en un Sistema Basado en el Conocimiento.

Habitualmente, en la tarea de diseño del clasificador, es necesario asumir ciertos compromisos o simplificaciones. En este sentido, y ya en un plano mucho más concreto, resulta de gran utilidad la suposición de normalidad para la forma de las distribuciones de probabilidad que siguen los datos pertenecientes a las distintas categorías presentes en un problema. También es una suposición frecuente en numerosos procedimientos estadísticos, bien sea por hacer numéricamente tratables los procesos o por simplicidad. Si bien, inicialmente, el estudio de la normalidad se trató como una propiedad de los datos que permitía crear clasificadores particularmente sencillos, la importancia que la distribución normal tiene en multitud de contextos tanto teóricos como prácticos, parecía justificar un estudio más profundo. Por este motivo, a medida que fue evolucionando el trabajo, el problema de la determinación de la normalidad de una muestra de datos fue adquiriendo mayor relevancia hasta convertirse en una parte muy importante.

En cualquier caso, es conveniente dejar claro que, tanto en el problema del diseño del Sistema Basado en el Conocimiento como en el estudio de la normalidad, las técnicas desarrolladas son generales y, por lo tanto, susceptibles de ser aplicadas a otro tipo de problemas. De hecho, aunque en el título del trabajo se hace referencia a dos aplicaciones concretas: patologías cerebrales y dislexias, cualquier problema de diagnóstico, planteado como un problema de clasificación de patrones, sería igualmente tratable.

Una vez expuestos los objetivos generales del trabajo, pasamos a comentar en más detalle algunos aspectos de los problemas tratados.

Las dificultades en el análisis, diseño e implementación de un Sistema Basado en el Conocimiento son enormes, mucho mayores que las de cualquier sistema software convencional. Con el fin de solucionar estas dificultades, surgen en la última década diversas metodologías como *Protégé-II*, *Generic Tasks*, *Components of Expertise* o *CommonKADS*, siendo esta última la elegida para la realización de este trabajo. Las razones de esta elección son su alto grado de refinamiento y su exhaustividad, cubriendo todos los aspectos que rodean al diseño de proyectos orientados al conocimiento.

Básicamente, estas metodologías pretenden simplificar el problema de la adquisición de conocimiento y propiciar la reutilización, en diferentes dominios, de las estrategias de razonamiento empleadas. Para que esto sea posible, es necesario separar estas estructuras de razonamiento del conocimiento sobre el que se aplican, manteniendo esta separación incluso en la fase de implementación. La metodología CommonKADS contempla estas situaciones a través de diferentes modelos, en especial el Modelo de Conocimiento y el Modelo de Diseño. Siguiendo estas premisas, nos planteamos el objetivo de implementar el Sistema Basado en el Conocimiento, sin perder el conjunto de estructuras establecidas en la fase de análisis. En términos del CommonKADS, esto significa, en definitiva, hacer "ejecutable" el Modelo de Conocimiento.

En lo que se refiere al diseño de clasificadores, nos hemos centrado en la clasificación supervisada, en la que se dispone de un conjunto de datos previamente clasificados que sirven para el diseño y validación del sistema. Hay que reiterar la dificultad del problema, debido a la gran incertidumbre que supone tener que basar todo el proceso en un conjunto de datos, generalmente escasos. Es realmente este factor, la carencia de información, lo que complica notablemente el diseño, ya que el abanico de posibles clasificadores se amplía de forma considerable.

Esta circunstancia suele conducir, en la práctica, a un procedimiento de búsqueda basado en prueba y error, ayudado por cierta intuición adquirida a partir de la visualización (cuando ésta sea posible) de los datos. Esta forma de resolver el problema no es demasiado satisfactoria y por eso se ha intentado un enfoque alternativo. Se trata de reducir el número de posibles diseños, haciendo explícito el conocimiento contenido en los datos disponibles y utilizando.

Por último, el estudio de la normalidad de las distribuciones de un conjunto de datos, suele abordarse como un contraste de hipótesis clásico, en el que sólo se especifica la hipótesis de normalidad, quedando las alternativas indeterminadas. Entre las críticas que, a menudo, se hacen a este tipo de tests, destaca la falta de un indicador fiable del grado de certeza de la hipótesis para unos datos concretos. En el contexto clásico, se intenta suplir esta carencia con la introducción del denominado valor-P, aunque su utilidad ha sido puesta en tela de juicio por numerosos investigadores.

En el contexto bayesiano este conflicto queda resuelto satisfactoriamente con la utilización de las probabilidades a posteriori. No resulta evidente la aplicación de la metodología bayesiana al problema anterior con una sola hipótesis. En este sentido, se han intentado dos enfoques alternativos, uno basado en redes neuronales, y el otro en Teoría de Grandes Desviaciones, que pretenden, desde un punto de vista más bayesiano, proporcionar procedimientos informativos y eficientes para la resolución de este problema.

La distribución por capítulos de esta memoria es la siguiente:

El capítulo 1 está dedicado a los fundamentos básicos de los Sistemas Basados en el Conocimiento y de la clasificación de patrones, que sirven como introducción al resto del trabajo.
En el capítulo 2 se explica el procedimiento seguido para implementar el Modelo de Conocimiento de la metodología CommonKADS.
En el capítulo 3 se plantea el problema general de la verificación de la normalidad de un conjunto de datos.
En el capítulo 4 se usan las técnicas desarrolladas en los capítulos 2 y 3, para la implementación de un Sistema Basado en el Conocimiento en el dominio del diseño de clasificadores. Al final del capítulo se aplica el sistema implementado a dos problemas concretos: el diagnóstico del Alzheimer y el diagnóstico de la Dislexia.

CAPÍTULO 1. FUNDAMENTOS

Este capítulo está dedicado a fundamentos básicos que sirven como introducción al resto del trabajo. Se divide en dos partes principales para dar cuenta de los dos grandes temas que se han tratado en esta memoria.

Por un lado, se introducen conceptos básicos relativos a la representación del conocimiento y como puede ser utilizado en la resolución de una determinada tarea. Se muestra la evolución de los Sistemas Expertos hacia los Sistemas Basados en el Conocimiento y se plantea la necesidad del uso de metodologías para su construcción. Finalmente, se introduce la metodología CommonKADS que se ha sido utilizada en el desarrollo de una parte importante del trabajo, como se describe en el capítulo 2.

Por otro lado, se plantea el problema de la clasificación de patrones y se da una breve descripción de algunos tipos de clasificadores. Se parte de la regla de Bayes y se introducen algunas definiciones fundamentales como la probabilidad promedio de error. Se sigue con una breve descripción de algunos de los modelos de clasificadores más comunes, como pueden ser los discriminantes lineal y cuadrático, las redes neuronales, clasificadores no paramétricos, árboles de decisión, ... Estos modelos constituyen la base para el diseño de un clasificador, asunto que se discute en profundidad en los capítulos 3 y 4 (especialmente en este último).

1.1 Fundamentos de Sistemas Basados en el Conocimiento

1.1.1 Importancia del Conocimiento

Los primeros desarrollos de la Inteligencia Artificial (IA) consistían en técnicas muy generales tales como búsquedas en espacios de estados que producían resultados espectaculares sobre pequeños problemas de prueba.. Estos métodos escalaban muy mal a los problemas reales, que poseen unos requerimientos computacionales muy elevados debido a la gran dimensionalidad de los espacios de búsqueda resultantes. La aplicación de heurísticas muy generales a las búsquedas alivia sólo ligeramente estos importantes problemas. Pronto quedó claramente demostrado la importancia del conocimiento en la resolución de problemas y por tanto la necesidad de representar adecuadamente la información sobre los mismos.

1.1.2 Representación del Conocimiento

El término “conocimiento” es fácil de comprender intuitivamente, pero difícil de definir de una manera formal. Lo que parece claro es que nos sirve para interpretar y manejar otras piezas de información, esto es, si un paciente tiene una temperatura de 39ºC, el

médico posee el conocimiento para concluir, a partir de esta información, que el paciente tiene fiebre.

Una de la cuestiones claves relacionadas con el conocimiento es su representación. Rich y Knight [Rich 91] [Russell 95] señalan cuatro características que debe verificar un sistema de representación del conocimiento

- Adecuación representacional

  Es la posibilidad de representar todos los tipos de conocimiento que aparezcan en el dominio del problema.

- Adecuación inferencial

  Es la habilidad de manipular las estructuras de conocimiento para derivar conocimiento nuevo a partir del nuevo.

- Eficiencia inferencial

  Consiste en la posibilidad de incorporar conocimiento adicional para restringir el proceso de inferencia en las direcciones más prometedoras.

- Eficiencia de adquisición

  Es la cualidad de incorporar nueva información de forma sencilla, ya sea manualmente por un ingeniero del conocimiento o bien de forma automática.

1.1.3 Ontologías Generales

Independientemente de los mecanismos de representación de conocimiento que se usen finalmente, existen varias clases de objetos abstractos que, por su generalidad, se emplearán muy frecuentemente en cualquier sistema. Estos objetos forman la base de una ontología general. Una *ontología* es un término tomado de la filosofía que trata sobre el ser, es decir, su objeto de estudio es lo que existe. En IA, lo que existe se reduce a lo que se representa en cada sistema. Así que el término ontología se usa como el estudio de los objetos que se van a representar y determinará el vocabulario o léxico a emplear en el sistema. Por tanto cada dominio de problema tendrá asociada una ontología elaborada en una fase de análisis del problema denominada conceptualización. Cabe preguntarse si sería posible determinar una ontología hábil para

representar una gran parte del universo. Los creadores del sistema CYC (CYC proviene de enciclopedia) intentan precisamente eso, elaborar una categorización de conceptos válida para representar conocimiento general y superar así la fragilidad de los sistemas actuales [Lenat 90]. La elaboración de ontologías es una actividad de diseño en ingeniería, y por tanto no admite una solución única. Aún así se reconocen ciertas clases de objetos abstractos que deberán considerarse en cualquier representación

- Categorías

  Se consigue una gran economía de expresión en la base de conocimientos organizando los objetos de características similares en categorías o clases. Estas categorías a su vez se introducen como objetos (clases) de pleno derecho en la base de conocimiento. Las propiedades o características compartidas por los objetos pertenecientes a la categoría son asociadas al objeto clase. De esta manera, simplemente afirmando la pertenencia de un objeto a la clase se pueden deducir muchas de sus características. Nada impide que se consideren clases cuyos miembros sean otras clases más particulares, estableciéndose relaciones entre clases, desde las más generales a las más concretas, constituyendo las llamadas taxonomías o clasificaciones jerárquicas de objetos.

- Se suelen considerar en este contexto fundamentalmente dos relaciones binarias importantes: ejemplar-de y es-un.

- Objetos Compuestos

  Muy frecuentemente, los objetos estarán constituidos por partes más simples que deben ser también consideradas como elementos del problema a resolver. Es importante entonces reflejar estas relaciones (“tener parte” y su inversa, “ser parte de”) explícitamente en la base de conocimientos.

- Evolución temporal: eventos y procesos

  La dependencia del tiempo de los fenómenos a analizar y la propia

representación del tiempo son cuestiones importantes. La distinción entre sucesos instantáneos o sucesos que se desarrollan a lo largo de un intervalo o bien entre eventos con un desarrollo en tiempo y espacio discreto o sucesos sin estructura interna denominados procesos.

- Objetos y eventos mentales

  Otra serie de conceptos útiles en representación, sobre todo en sistemas con múltiples agentes son los objetos y eventos mentales. Se suele recurrir a adscribir intenciones y estados mentales a otros agentes que interaccionan en el problema. Serán objeto de razonamiento las posibles intenciones de otros agentes o cualquier otro tipo de sucesos mentales (creencias, etc.).

1.1.4 Mecanismos de Representación

En este apartado se verán una serie de posibilidades que se pueden emplear en la representación de conocimiento. Volvemos a considerar estas representaciones como lenguajes, a pesar de que algunas representaciones sean de tipo gráfico, a base de arcos dirigidos y cajas. En definitiva, también constituyen un lenguaje del que expondremos su sintaxis y semántica. La representación del conocimiento en forma de reglas se tratará en un apartado posterior.

1.1.4.1 Lógica

La lógica, debido a su sólida estructura como representación del conocimiento juega el papel de estándar con el que se comparan otros métodos de representación. Así, se compara la expresividad de un método equiparándolo con lógica de predicados, de primer orden etc. Se han elaborado varios formalismos en lógica de primer orden para elaborar alguna de las categorías de objetos anteriores. Como ejemplo, el cálculo de situaciones propuesto por McCarthy para el problema de la dependencia del tiempo y las posibles evoluciones del estado del problema [McCarthy 60].

La lógica de predicados de primer orden no está exenta de limitaciones. La característica de *monotonicidad*, que implica que el conjunto de hechos deducidos de

una teoría siempre aumenta, impide la revisión del conocimiento. Para evitar estos problemas se ha extendido la lógica en múltiples direcciones, lo que provoca por otro lado un gran aumento en la complejidad computacional de los procesos de inferencia en estas lógicas extendidas. Una de estas extensiones no monótonas es el *razonamiento por defecto*, que permite describir un estado del mundo "por defecto", que en ausencia de información en sentido contrario se asume como cierta.

1.1.4.2 Representaciones de Conocimiento con Estructura.

Los lenguajes formales basados en la lógica tienen en la sentencia su unidad fundamental. Cada sentencia expresa por sí misma una porción de conocimiento independientemente de las demás. Un conjunto de sentencias no necesita ser construido en ningún orden especial y todas las sentencias se consideran en pie de igualdad. De esta manera, se logra representar la información sin considerar las posibles formas en las que ese conocimiento se puede utilizar. Se dice entonces que se está usando una representación declarativa, en contraste con la representación procedural. En ésta última el conocimiento se entremezcla con mecanismos de control rígidos que especifican una de entre las múltiples formas en que es usado para resolver un problema. Sin embargo, esta cualidad de los lenguajes lógicos puede también oscurecer la comprensión del cuerpo de conocimiento almacenado en un conjunto de sentencias, ya que no permite apreciar la estructura del mismo. En esta sección se presentarán varios métodos de representación que incorporan mecanismos para hacer patente esa estructura. Algunos de ellos se apoyan en construcciones gráficas que ayudan a la visualización de las relaciones entre conceptos. La mayor parte de estos sistemas se centran en la representación de categorías de conceptos, a través de relaciones *subconjunto-de* que se ordenan jerárquicamente y *miembro-de*.

Redes Semánticas

Las redes semánticas son colecciones de nodos conectados entre sí por arcos con dirección (grafo dirigido). Fueron propuestas inicialmente por M. R. Quillian. Los nodos representan conceptos o entidades y los arcos representan relaciones binarias que se verifican entre los dos nodos que unen y con la dirección indicada. Cada arco tiene el

nombre de la relación binaria que representa. En la figura 1.1 aparece un ejemplo sencillo de red.

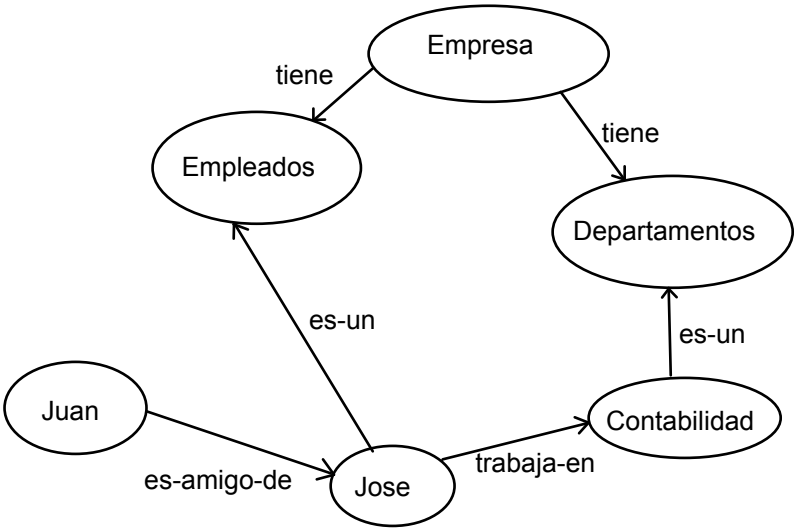

Figura 1.1. Red Semántica

Las relaciones más comunes son las que expresan la categorización de los objetos:

- *subconjunto-de* o *es* para indicar la inclusión de un subconjunto en un conjunto o categoría superior.
- *miembro-de* o ∈ (*instance-of*) para indicar la pertenencia del elemento a un conjunto o clase.

Las relaciones entre más de dos elementos se pueden realizar creando una nueva entidad (nodo) que interconecte esos elementos. Es importante especificar la semántica del formalismo, en particular, para aprovechar la noción de *herencia*, en la cual un objeto adquiere las propiedades características de la clase o clases a las que pertenece sin necesidad de incluirlas explícitamente como elementos del grafo. Esto requiere indicar las uniones que se verifican entre objetos sencillos, las uniones que se verifican entre objetos clase, y en este último caso si éstas relaciones se extienden a todos los objetos miembros de la clase o sólo son características de la clase como un todo. La *herencia múltiple* consiste en la existencia de varios posibles caminos de herencia cuando un objeto pertenece a varias clases "padre".

Se ha demostrado la equivalencia entre las redes semánticas y un tipo restringido de lógica de relaciones binarias. Algunos autores han extendido el formalismo hasta hacerlo equivalente en expresividad a la lógica de primer orden.

Marcos o *Frames*

El marco o *frame* es otro tipo de estructura que permite expresar conceptos ligándolos con las propiedades que poseen. Cada marco representa un concepto y está compuesto de una serie de "ranuras" (*slots*) con nombre que representan sus propiedades. Los slots contienen valores para estas propiedades que pueden incluir apuntadores a otros marcos. Un marco puede representar una clase genérica de objetos o bien una instancia o ejemplar concreto. En este último caso se dice que es un marco instanciado. Una misma entidad puede ser considerada desde varios puntos de vista diferentes, cada uno recogido en un marco, simultáneamente.

Ciertos valores del marco que representa una clase pueden estar ya inicializados a valores concretos. Estos valores son propiedades típicas de la clase, en el sentido de que todos los miembros de la misma deben verificarlas. Se conocen como propiedades genéricas y, en conjunto, forman la definición del objeto.

Otro tipo de valores que es posible encontrar en un marco asociado a una clase son los valores *por defecto*. En este caso no es obligatorio que los miembros de la clase presenten dicho valor, pero en ausencia de información más específica el dato puede ser utilizado.

Las ranuras pueden contener condiciones asociadas que limitan el tipo de valores que pueden acoger. Como ejemplos de estas condiciones se tienen rangos de validez, cardinalidad o número de posibles contenidos en la ranura.

En muchos sistemas basados en marcos se complementa esta representación uniendo a las ranuras rutinas procedurales que son accionadas bajo ciertas condiciones. Por ejemplo, para que se realice el cálculo del *slot Área* en un marco asociado a un rectángulo en cuanto cambien los *slots Longitud-Base* o *Longitud-Altura*.

Las inferencias más comunes consisten en la identificación de un determinado objeto desconocido mediante el pareamiento de sus *slots* con los *slots* genéricos definidos en los marcos. Otro tipo de inferencia es la deducción de valores por defecto heredados para *slots* desconocidos.

1.1.5 Sistemas Basados en Reglas (Sistemas Expertos Tradicionales)

Uno de los grandes éxitos de la IA reciente ha sido el desarrollo de los llamados Sistemas Expertos (SE). Estos sistemas introducen gran cantidad de conocimiento sobre

un campo o dominio limitado, pudiendo lograr un grado de competencia comparable al del experto humano en dicho campo. El mayor interés de estos sistemas está en hacer disponible el conocimiento de estos expertos, que suelen ser muy escasos y cuyo tiempo es muy valioso, en forma de un programa. De esta forma el sistema se utiliza como depósito de dicho conocimiento, accesible a múltiples usuarios, liberando también de trabajo a los expertos para dedicarse a los problemas más difíciles.

Entre las ventajas e inconvenientes de los Sistemas Expertos, podemos enumerar las siguientes:

Ventajas:

- Permiten recoger y distribuir el conocimiento del experto en la organización
- Hacen explícito el conocimiento, permitiendo la formalización de la experiencia
- Son útiles en la educación de nuevos profesionales

Inconvenientes:

- No poseen conocimiento general, de "sentido común"
- El tratar con incertidumbre es difícil
- Su construcción es muy costosa, debido al problema de la "adquisición del conocimiento"
- No poseen mecanismos de aprendizaje

Las reglas o producciones son la unidad de representación de conocimiento básica en este tipo de sistemas. Se corresponden con locuciones tipo

SI P ENTONCES C

Donde *P* es una conjunción de condiciones o premisas y *C* es un conjunto de conclusiones o acciones que se verifican o ejecutan si las condiciones son todas ciertas.

```
Si
       el paciente tose y tiene fiebre
entonces
       el diagnóstico es gripe
Si  los conectores son de igual nº de pines, y
       el bus acaba en un terminador, y
       el periférico no está alimentado
Entonces
       conectar el periférico al bus
```

Los expertos encuentran más natural expresar su conocimiento en forma de reglas sencillas que en otros formalismos más potentes y complejos, como la lógica de primer orden.

Este tipo de sistemas se conoce también como sistemas de producción y se suelen emplear para denotar las normas de escritura en la descripción de gramáticas formales o de reconocedores de sintaxis.

En general, un sistema de producción está compuesto por los elementos que se muestran en la figura 1.2.

- Base de Hechos
- Base de Reglas
- Sistema de Control o motor de inferencia.

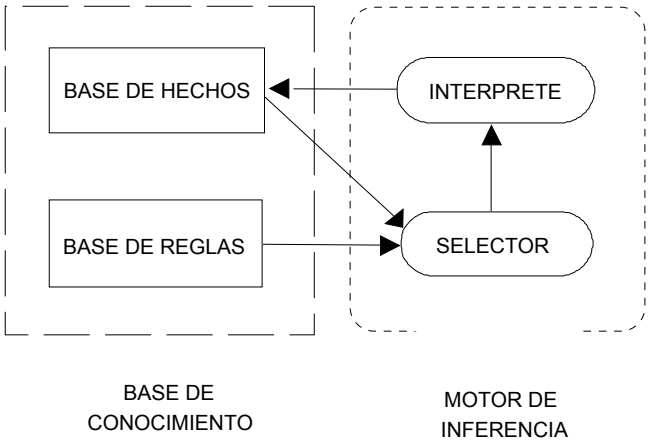

Figura 1.2. Esquema de un Sistema de Producción

La Base de Hechos contiene inicialmente los datos concretos del problema que se va a resolver. Los hechos son afirmaciones relevantes al problema (referentes a objetos que forman parte del dominio del problema), usualmente en forma de pares <objeto, valor> o tripletas <objeto, atributo, valor>. La parte de las premisas o condiciones de las reglas se refieren a estos hechos. Si estas condiciones se verifican en el estado actual de la base de hechos, se deducen los hechos que figuran en las conclusiones de la regla. Las conclusiones de las reglas que se han verificado pasan también a la base de hechos, contribuyendo a satisfacer nuevas reglas.

La Base de Reglas está formada por el conjunto de reglas usadas en la resolución del problema. Si las conclusiones de las reglas sólo contienen nuevas afirmaciones para introducir en la base de datos, estamos ante un sistema puramente deductivo. Si las conclusiones especifican acciones como ejecutar el código de una subrutina o ejercer algún cambio sobre el sistema (con o sin efecto sobre la base de hechos) se le llama reactivo. En general se encuentra una gran variedad de diferencias en los distintos sistemas que afectan a la forma de regla permitida: la composición lógica de antecedentes y conclusión, si se permite el uso de variables en la regla y qué tipo de cuantificación les afecta, etc. Esta gran variabilidad permite que en ocasiones la semántica de las reglas no esté completamente clara, y se den efectos indeseables no previstos por el implementador del sistema.

El Sistema de Control es el responsable de elegir y aplicar las reglas que puedan provocar cambios en la base de datos. Estos cambios darán lugar al accionamiento de otras reglas hasta llegar a una conclusión final del problema o a un estado de parada en el que ninguna regla puede verificarse.

Existen múltiples posibilidades para establecer la selección de reglas, pero fundamentalmente hay dos enfoques principales: el basado en los datos o encadenamiento progresivo y el basado en objetivos o encadenamiento regresivo.

En el encadenamiento progresivo, los hechos de la base de conocimiento se comparan con el antecedente de la regla. Si concuerdan, los hechos que aparecen en el consecuente de la regla entran en la base de conocimiento. Esto es el “disparo” de la regla

En el encadenamiento regresivo, dada una regla, se toma como hipótesis su consecuente y se intenta encontrar hechos u otras reglas cuyo consecuente concuerden con su antecedente, en un proceso “hacia atrás”

1.1.5.1 Herramientas para el Desarrollo de SE: *Shells*

La herramienta por excelencia para la implementación del Sistema Experto es la *Shell*. Una *shell* consiste en un sistema experto cuya base de conocimientos es creada por el usuario. El ingeniero de conocimiento elabora la base de conocimiento formalizando el conocimiento aportado por un(os) experto(s) en un dominio particular. El tipo de control de inferencia viene prefijado en la herramienta, así como la estructura permitida

de las reglas. Las *shells* más evolucionadas permiten varios formalismos para la representación del conocimiento, además de las reglas. Proporcionan además ayudas a la elaboración de la base de conocimiento, comprobación de consistencia, medios para estructurar las reglas, y facilidades de depuración como, por ejemplo, mecanismos de explicación de conclusiones. En algunos casos, también incluyen facilidades para la creación de interfaces de usuario o integración con lenguajes tradicionales de programación.

1.1.5.2 El Problema de la Adquisición del Conocimiento en los Sistemas Expertos

Un problema fundamental en la construcción de un Sistema Experto es la adquisición del conocimiento. En los siguientes puntos se resumen las principales dificultades relacionadas con esta cuestión.

- Las bases de conocimiento resultan ser inmanejables debido a la falta de estructura.
- El desarrollo, debido a lo abierto del proceso, no puede ser planificado adecuadamente.
- Los mecanismos de representación del conocimiento uniformes (reglas), tienden a ocultar detalles importantes del proceso de razonamiento.
- La oportunidad de reusar el conocimiento para resolver otros problemas es mínima.

Para dar cuenta de estas dificultades surgen, en la década de los 90, una serie de metodologías, a las cuales nos referiremos en más detalle en la sección siguiente.

En lo que sigue, sustituiremos la noción de Sistema Experto por la de Sistema Basado en el Conocimiento (SBC). Realmente, se trata de una evolución de la denominación original, más moderada y dejando patente la importancia que el conocimiento tiene para la resolución de problemas. Se puede ver, además, como una generalización del Sistema Experto tradicional, "encorsetado" en el formalismo de reglas, permitiendo una mayor flexibilidad en la representación y utilización del conocimiento.

1.1.6 Metodologías para el Diseño de Sistemas Basados en el Conocimiento

El diseño de un Sistema Basado en el Conocimiento dista de ser una tarea trivial. El coste de implementación de un sistema de este tipo excede con mucho al esfuerzo de realizar una aplicación informática convencional. Las metodologías para el diseño de SBCs nacen con la idea de construir sistemas de calidad, a gran escala, estructurados, controlables y fácilmente replicables. Los principios que sustentan estas metodologías pueden resumirse en los siguientes puntos:

- Cambia el énfasis del "cómo" al "por qué", desligando el análisis de las estrategias del experto de mecanismos de implementación concretos.
- El análisis y diseño de SBCs son actividades de modelización de los procesos de resolución de problemas del experto.
- Se distinguen distintos tipos de conocimiento, para permitir su reutilización.
- Un proyecto de Ingeniería del Conocimiento debe ser llevado a cabo "sobre la marcha", de una forma controlada, siguiendo un desarrollo en espiral [Boehm 88].

Se trata, en definitiva, de principios parecidos a los que se utilizan en la Ingeniería de Software convencional y persiguen objetivos similares.

El primero de los puntos tiene como referencia fundamental el Principio del Nivel de Conocimiento (*Knowledge-Level Hypothesis*) de Alan Newell [Newell 82], y que viene a decir que en el modelado del conocimiento hay que concentrarse primero en la estructura conceptual y dejar los detalles y mecanismos de implementación para más tarde.

El segundo de los puntos se refiere a la conveniencia de construir modelos de los diferentes aspectos del conocimiento, permitiendo centrarse en aquellos que sean de interés en cada momento.

El tercer punto también es de gran importancia, ya que permitiría soslayar el cuello de botella de la adquisición de conocimiento. El diseño de técnicas y principios básicos para el desarrollo de bases de conocimiento de forma que puedan ser reusadas y

compartidas fuera del ámbito para el que fueron originalmente pensadas es uno de los principales temas de investigación en Ingeniería del Conocimiento. El principio general que determina la reusabilidad es que se puedan diferenciar tipos de conocimiento independientes y distintos de acuerdo a su función. Quizás la distinción más inmediata es la que se puede hacer entre conocimiento del dominio y conocimiento de control. El primero es de tipo estático, declarativo y está compuesto de los conceptos, relaciones y hechos que son necesarios para el razonamiento sobre un dominio concreto. El segundo describe como se realiza el razonamiento en términos de operaciones de razonamiento elementales sobre elementos del dominio y también en términos de las estructuras de control y descomposiciones de la tarea global. Este segundo tipo de conocimiento es de naturaleza más procedural. Aquí podemos encontrar los llamados métodos de resolución de problemas genéricos. Desafortunadamente, estos dos tipos de conocimiento no son independientes. Chandrasekaran formuló la hipótesis de interacción [Chandrasekaran 88], que afirma que ambos tipos de conocimiento están altamente interrelacionados de forma que no se puede definir el conocimiento del dominio sin conocer la forma en que va a ser usado (control) ni a la inversa. A pesar de todo, se ha intentado atenuar el problema distinguiendo varios tipos de interacción y haciéndolas explícitas.

El cuarto punto propone una aproximación al problema siguiendo un desarrollo en espiral, frente a otras posibilidades como el modelo en cascada (*waterfall*) o el prototipado rápido (*rapid prototyping*) [Sommerville 95]. De esta manera, se consigue más flexibilidad y control, utilizando resultados o estados intermedios de los modelos como indicadores de lo que hay que hacer a continuación.

Estos puntos generales se hacen operativos en metodologías como *CommonKADS* (*Common Knowledge Analysis and Design Support*) [Schreiber 94] [Schreiber 99], la cual evolucionó a partir de KADS [Wielinga 92] y *Components of Expertise* [Steels 90]; y otras metodologías desarrolladas en los Estados Unidos, tales como *Generic Tasks* [Chandrasekaran 93], *PROTÉGÉ-II* [Tu 95], y *Role-Limiting Methods* [Marcus 88].

En este trabajo nos hemos centrado en la metodología CommonKADS, y a ella está dedicada la sección siguiente.

1.1.7 Fundamentos de la Metodología CommonKADS

Los orígenes de la metodología CommonKADS se remontan al año 1983, aproximadamente. En un principio se denominó KADS, y es a partir de 1995 cuando se empieza a hablar de CommonKADS.

Se fundamenta en los principios comentados en la sección anterior, y su expresión práctica queda reflejada en los diferentes modelos representados en la figura 1.3.

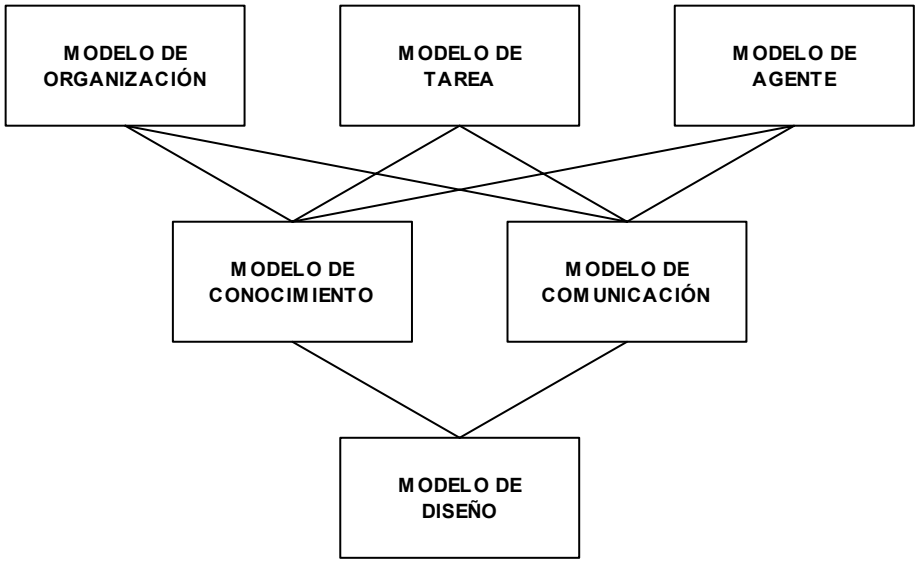

Figura 1.3. Modelos de la Metodología CommonKADS

Cada uno de estos modelos se enfoca hacia un aspecto determinado de la Ingeniería del Conocimiento. A continuación se describe brevemente cada uno de ellos.

Modelo de Organización

El Modelo de Organización se ocupa de las cuestiones principales relacionadas con la organización, desde la perspectiva de la integración del sistema en un contexto determinado.

Modelo de Tarea

El Modelo de Tarea analiza la tarea global a desempeñar por el sistema, sus entradas y salidas, precondiciones y criterios de rendimiento, así como los recursos que se necesitan.

Modelo de Agente

Los agentes son los ejecutores de una tarea. El Modelo de Agente describe las características de un agente, en particular, sus competencias, autoridad para actuar y restricciones.

Modelo de Conocimiento

El propósito del modelo de conocimiento es describir, en detalle, los tipos y estructuras de conocimiento utilizados en la realización de una tarea. Esta descripción es independiente de la implementación e indica que papel juegan los diferentes componentes del conocimiento, de manera inteligible para una persona. Esto convierte al Modelo de Conocimiento en un vehículo importante para la comunicación con los expertos humanos y usuarios, durante el proceso de desarrollo y ejecución del sistema.

Modelo de Comunicación

Como varios agentes pueden estar involucrados en una tarea, es importante modelar las transferencias de información que existen entre dichos agentes. Esta es la función del Modelo de Comunicación, y al igual que para el Modelo de Conocimiento, esto se realiza de una forma conceptual e independiente de la implementación.

Modelo de Diseño

Los modelos anteriores de CommonKADS pueden considerarse como una especificación de los requisitos que debe satisfacer el Sistema Basado en el Conocimiento. Basándose en estos requisitos, el Modelo de Diseño aporta la especificación técnica, en términos de arquitectura software, plataforma de implementación, módulos software, mecanismos de representación y computacionales, necesarios para implementar las estructuras de conocimiento descritas en los Modelos de Conocimiento y de Comunicación.

En el próximo capítulo nos ocuparemos ampliamente de dos de los modelos expuestos: el Modelo de Conocimiento y el Modelo de Diseño.

1.1.8 Sistemas Expertos Probabilísticos. Redes de Creencia Bayesianas

En esta sección incluimos una breve descripción de una alternativa a los Sistemas Expertos tradicionales que proporciona mecanismos muy potentes para representar y propagar la incertidumbre. Por incertidumbre se entiende la incapacidad de atribuir un valor de verdad o falsedad a un hecho, debido a la ignorancia o a la complejidad del dominio.

Los Sistemas Expertos tradicionales introdujeron construcciones *ad hoc* como los factores de confianza (*confidence factors*) que permitían combinar el grado de creencia de los antecedentes para obtener un grado de creencia en el consecuente. Muchos de estos mecanismos presentaban serios problemas, pudiendo llegar a ser inconsistentes [Weiss 84].

Uno de los mecanismos bien fundados para tratar con la incertidumbre es la probabilidad, en su interpretación *subjetiva*, de *grado de creencia* o Bayesiana. Así, se asigna a cada hecho una determinada probabilidad, por ejemplo:

P(Dolor_de_Cabeza)=0.2.

De esta manera, una descripción completa probabilística de un dominio significa asignar a cada posible combinación de hechos, una probabilidad, esto es, para *n* hechos, 2^n probabilidades. Veamos el siguiente ejemplo con n=3.

C="Dolor_de_Cabeza", L="Hipersensibilidad_a_la_luz", D="Migrañas"

Las 2^n=8 probabilidades serían en este caso:

P(D,C,L), P(noD,C,L), P(D, noC, L), P(noD, noC,L), P(D,C,noL), P(noD,C,noL), P(D, noC, noL), P(noD, noC,noL).

La regla que permite actualizar las probabilidades a la luz de las evidencias es la regla de Bayes:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

donde P(A|B) indica la probabilidad del hecho A supuesto que se ha dado B (en secciones posteriores se volverá a mencionar la regla de Bayes y se explicará con mayor detalle).

Las redes de creencia o causales [Lauritzen 88] [Pearl 88] son representaciones gráficas que permiten describir de forma visual un modelo probabilístico, como muestra el ejemplo de la figura 1.4.

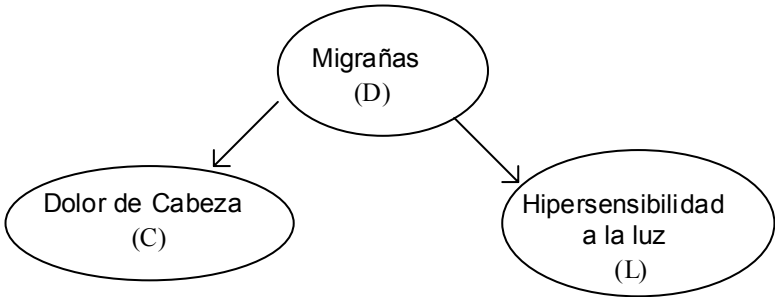

Figura 1.4. Ejemplo de Red de Creencia

Estas redes de creencia poseen las siguientes características generales:

- Si dos nodos no están conectados directamente, son condicionalmente independientes: P(C|D,L)=P(C|D), o P(L|D,C)=P(L|D)
- Constituyen una alternativa a las reglas, con un formalismo riguroso para propagar incertidumbre
- Los arcos entre los nodos representan relaciones de causalidad o relevancia
- Cada nodo al que llegue un arco debe tener asociada las probabilidades condicionales de sus valores respecto los nodos de partida del arco
- Los nodos que no son apuntados por ningún otro contienen una probabilidad a priori (probabilidad antes de observar ninguna evidencia)
- Una red permite incorporar evidencia y actualizar la probabilidad de cada nodo correctamente

A pesar de que se trata de una técnica bien establecida y formalizada, no está exenta de problemas. En primer lugar, la forma de representar el conocimiento está bastante alejada de la forma natural en la que un experto expresa su grado de creencia. Por otro lado, es necesario disponer de un gran número de casos para poder estimar las probabilidades presentes.

1.2 Fundamentos de Clasificación de Patrones

1.2.1 Definición del Problema

La clasificación es la asignación de objetos a clases predeterminadas, en base a sus características. Estas características pueden ser de tipo cuantitativo, como, por ejemplo, un conjunto de potencias espectrales, o de tipo cualitativo, p. ej. la forma de un trazo electroencefalográfico. Al conjunto de estas características se le suele denominar patrón. Por lo tanto, la dificultad principal del problema estriba en la capacidad discriminatoria entre clases, del conjunto de características elegido.

Habitualmente, la naturaleza del problema es aleatoria, ante la imposibilidad de definir un modelo determinista de las clases. Para dar cuenta de esta aleatoriedad, las clases se caracterizan por medio de las densidades de probabilidad multivariantes de las características observadas. Este enfoque del problema de la clasificación, permite abordarlo desde la perspectiva de la Teoría Estadística de la Decisión, y será el contexto en el que desarrollaremos el resto del trabajo.

Teniendo en cuenta lo anterior, podemos definir, más formalmente, el problema de la clasificación de patrones como:

Dada de una serie de clases predefinidas $(C_1, C_2, \ldots, C_L)$, y un objeto caracterizado por un conjunto de medidas o características que dan lugar a un vector $X = (X_1, X_2, \ldots, X_n)$, la clasificación del objeto para el valor observado $X = x$, supone optar entre $L + 2$ decisiones. Estas decisiones pueden ser, o bien, asignarlo a una de las L clases, o bien, clasificarlo como "dudoso", posponiendo la decisión hasta tener mayor seguridad, o bien, clasificarlo como *outlier*, esto es, que no pertenece a ninguna de las clases.

Hay que aclarar que esta definición de clasificación se refiere, más bien, a lo que se conoce como clasificación supervisada, en la que las clases están predefinidas y se dispone de patrones previamente clasificados. A estos patrones, se les denomina conjunto de entrenamiento, y son la base para el diseño del clasificador. En la clasificación no supervisada, no se conocen las clases a priori y, en esencia, se trata de descubrir agrupamientos de patrones (nuevas clases), de acuerdo con algún criterio establecido. En cualquier caso, en este trabajo nos centraremos exclusivamente en la clasificación supervisada.

Si la densidad de probabilidad, condicionada a las clases, del vector de características es conocida, el problema de la clasificación de patrones se convierte en un problema de contraste de hipótesis estadístico que suele tratarse en el contexto de la Teoría Bayesiana de la Decisión. Si bien, el supuesto de densidades de probabilidad conocidas es muy improbable en la práctica, resulta interesante su consideración, ya que proporciona una marco para tratar el problema y establece límites a las prestaciones de un clasificador.

1.2.2 Teoría Bayesiana de la Decisión

La Teoría Bayesiana de la Decisión proporciona un marco adecuado para tratar el problema de la clasificación. El resultado fundamental, en el se basa el planteamiento posterior, es la conocida Regla de Bayes, que introducimos a continuación.

Regla de Bayes

La probabilidad condicional $p(a|b)$ se define como la probabilidad del suceso *a* si se ha dado el suceso *b*. Podemos expresar la probabilidad conjunta $p(a \cap b)$ de dos sucesos *a* y *b* (probabilidad de que se den el suceso *a* y el suceso *b*) como la probabilidad de que

se verifique el suceso *a* por la probabilidad de que se verifique *b* condicionado a *a* $p(a \cap b) = p(a)p(b|a)$, análogamente también se puede escribir como $p(a \cap b) = p(b)p(a|b)$. Si eliminamos $p(a \cap b)$ entre las dos expresiones obtenemos la *regla de Bayes*

$$p(a|b) = \frac{p(a)p(b|a)}{p(b)} \tag{1.1}$$

Podemos aplicar esta regla al problema de determinar la pertenencia de un objeto a una de dos clases C_1 y C_2. Inicialmente podemos suponer conocidas las probabilidades de pertenencia del objeto a las clases con independencia del valor de la observación de sus propiedades. Con estas probabilidades a priori $P(C_1)$ y $P(C_2)$, podemos tomar como regla de decisión el elegir la clase de mayor probabilidad.

Alternativamente, podemos usar la información de las propiedades observadas del objeto para obtener una decisión más fiable a través de la regla de Bayes, junto con el criterio de máxima verosimilitud. Esto es, conocidos $p(x|C_1)$ y $p(x|C_2)$, se aplicaría la regla de Bayes, para obtener:

$$p(C_i|x) = \frac{P(C_i)p(x|C_i)}{p(x)}, \quad i = 1, 2 \tag{1.2}$$

donde $p(x)$ es la probabilidad total (independientemente de las clases) de una observación de valores x, y $P(C_i|x)$ es la probabilidad a posteriori de la clase C_i.

Ahora, la regla de decisión quedaría como:

$$\begin{aligned} &x \in C_1 \text{ si } P(C_1|x) > p(C_2|x) \\ &x \in C_2 \text{ si } p(C_2|x) > p(C_1|x) \end{aligned} \tag{1.3}$$

Una vez establecida la regla de decisión, sería interesante disponer de algún criterio que diera cuenta de la eficiencia de la misma. En este sentido, el criterio más usado en el contexto de la clasificación de patrones es la probabilidad promedio de error, definida como:

$$P(error) = \int P(error|x)p(x)dx \tag{1.4}$$

Con la regla de decisión anterior, tenemos que $P(error \mid x)$ será igual a $P(C_1 \mid x)$, si se decide C_2, o $P(C_2 \mid x)$, si se decide C_1. Es muy fácil comprobar, que de esta manera se minimiza $P(error)$, y por lo tanto, se habría tomado la mejor decisión posible. Esto supone, que si tomamos este criterio de error como referencia, la aplicación de la regla de Bayes establece un límite en la eficiencia alcanzable por un clasificador, para unas características X dadas.

En la expresión anterior de la probabilidad promedio de error, se ha asumido que todos los errores son igualmente costosos, esto es, equivocarse al asignar a la clase 1 un objeto que pertenece a la clase 2, es equivalente a equivocarse en el otro sentido. En la práctica, a veces, resulta conveniente pesar estos errores de forma distinta. En estos casos, se plantea como criterio de bondad en la clasificación, una función de riesgo más general que tendría a la probabilidad promedio de error como caso particular.

A partir de lo expuesto, es posible introducir también el concepto de *función discriminante*, que resulta de gran utilidad como base para muchos de los métodos de diseño de clasificadores.

Así, se definen como un conjunto de funciones $D_i(x)$, cada una asociada a una clase C_i, de manera que el clasificador asigna el objeto a la clase *i* que verifica:

$$D_i(x) > D_j(x) \quad \forall j \neq i \tag{1.5}$$

Las funciones $D_i(x)$ son totalmente generales, no tienen que estar basadas necesariamente en argumentos probabilísticos. Desde el punto de vista geométrico, estas funciones dividen el espacio de las características en dos tipos de regiones. Por un lado, las regiones en las que todos los puntos verifican la condición anterior y por tanto pertenecen a una misma clase. Por otro lado, las regiones denominadas indeterminadas, en las que no se verifica la regla y por ello no es posible hacerles corresponder una única clase. Los límites entre las regiones vendrán determinados por las fronteras o superficies de decisión entre cada dos clases i y j:

$$S_{ij}(x) = D_i(x) - D_j(x) = 0 \tag{1.6}$$

En el caso de una decisión entre dos clases sólo hay una superficie de decisión $S_{12}(x) = 0$ y por tanto no quedan regiones indeterminadas: un punto pertenece a una clase u otra según quede a un lado u otro de la superficie de decisión.

La probabilidad a posteriori, antes introducida, puede ser considerada como una función discriminante, ya que

$$\begin{aligned} D_1(x) &= P(C_1 \mid x) \\ D_2(\mathbf{x}) &= P(C_2 \mid x) \end{aligned} \tag{1.7}$$

La frontera de decisión quedaría entonces, como:

$$\begin{aligned} S_{12}(x) &= 0 \\ P(C_1 \mid x) - P(C_2 \mid x) &= 0 \\ \frac{P(C_1)p(x \mid C_1)}{p(x)} - \frac{P(C_2)p(x \mid C_2)}{p(x)} &= 0 \\ P(C_1)p(x \mid C_1) - P(C_2)p(x \mid C_2) &= 0 \end{aligned} \tag{1.8}$$

expresión que se puede simplificar aún más en el caso de que las prioridades a priori de las dos clases sean iguales.

La forma analítica de la superficie de decisión depende de las distribuciones asumidas en cada clase $p(x \mid C_i)$. El caso más común consiste en aproximar las distribuciones reales de cada clase por distribuciones normales multivariantes. Cada clase vendrá caracterizada en la distribución por su vector de medias y su matriz de covarianza. En el caso de que las matrices de covarianza sean iguales en ambas clases, la superficie de decisión resulta tener una dependencia lineal con las componentes del vector característico x, siendo este caso conocido como discriminante lineal. La frontera de discriminación vendría dada, entonces, por:

$$(\mu_2 - \mu_1)^T \Sigma^{-1} x + \frac{1}{2}\left(\mu_1{}^T \Sigma^{-1} \mu_1 - \mu_2{}^T \Sigma^{-1} \mu_2\right) - \log \frac{P(C_1)}{P(C_2)} = 0 \tag{1.9}$$

donde μ_1 y μ_2 son los vectores de medias de las dos clases, y $\Sigma = \Sigma_1 = \Sigma_2$ es la matriz de covarianza.

En el caso en que estas matrices sean distintas, la dependencia con las componentes de x es cuadrática. Esta es la superficie de decisión más compleja a la que

se puede llegar, asumiendo normalidad. Este tipo de fronteras se conoce como discriminante cuadrático, y su ecuación es:

$$\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) - \frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) + \frac{1}{2}\log\frac{|\Sigma_1|}{|\Sigma_2|} - \log\frac{P(C_1)}{P(C_2)} = 0 \tag{1.10}$$

En la figura 1.4 se muestra un ejemplo de estas dos situaciones.

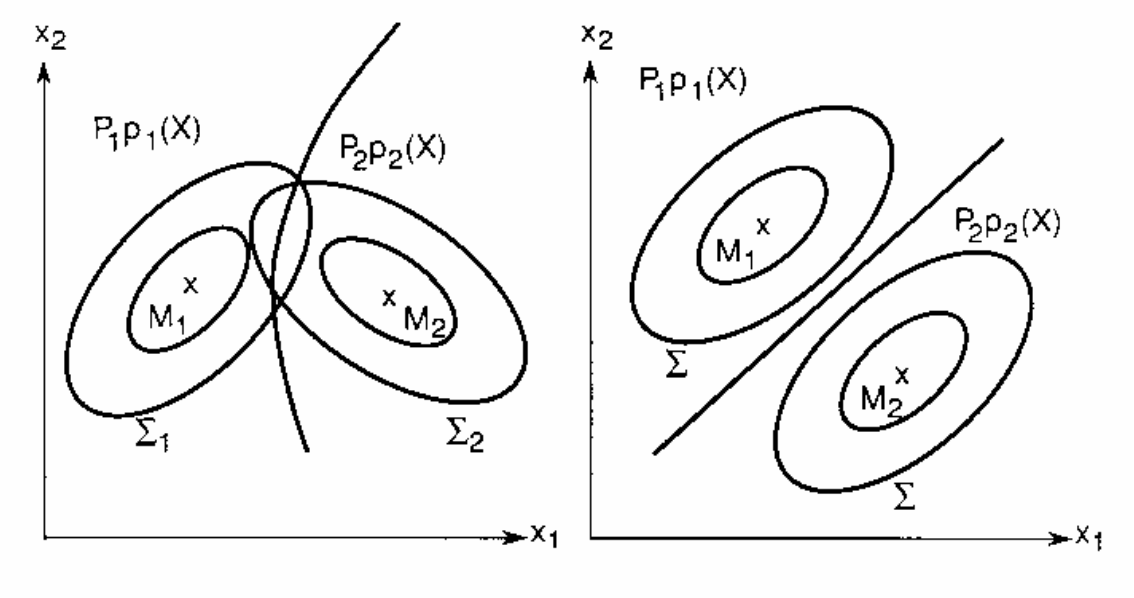

Figura 1.4. Superficies discriminantes para el caso cuadrático (izquierda) y el caso lineal (derecha), con distribuciones de clase normales (figura extraída de [Fukunaga 90])

Para concluir esta sección, simplemente aclarar que aunque se ha utilizado siempre el ejemplo de dos clases, todo lo dicho es extensible a múltiples clases.

1.2.3 Tipos de Clasificadores

En la discusión anterior, se hizo referencia a dos tipos de discriminantes obtenidos a partir de la suposición de normalidad en las distribuciones de probabilidad de las clases: el discriminante lineal y el discriminante cuadrático.

Existen, por supuesto, otros tipos de clasificadores, cada uno con sus especificaciones y problemática particular. Aunque hemos visto que calculando las probabilidades a posteriori, seremos capaces de conseguir la clasificación “ideal”, en la

práctica, diversos factores complican tremendamente este cálculo y se hace preciso abordar el problema desde otra perspectiva. Lo más habitual es ir probando diferentes modelos de clasificadores hasta dar con el más adecuado. En la figura 1.5 se muestra un esquema con algunos de los clasificadores más comunes.

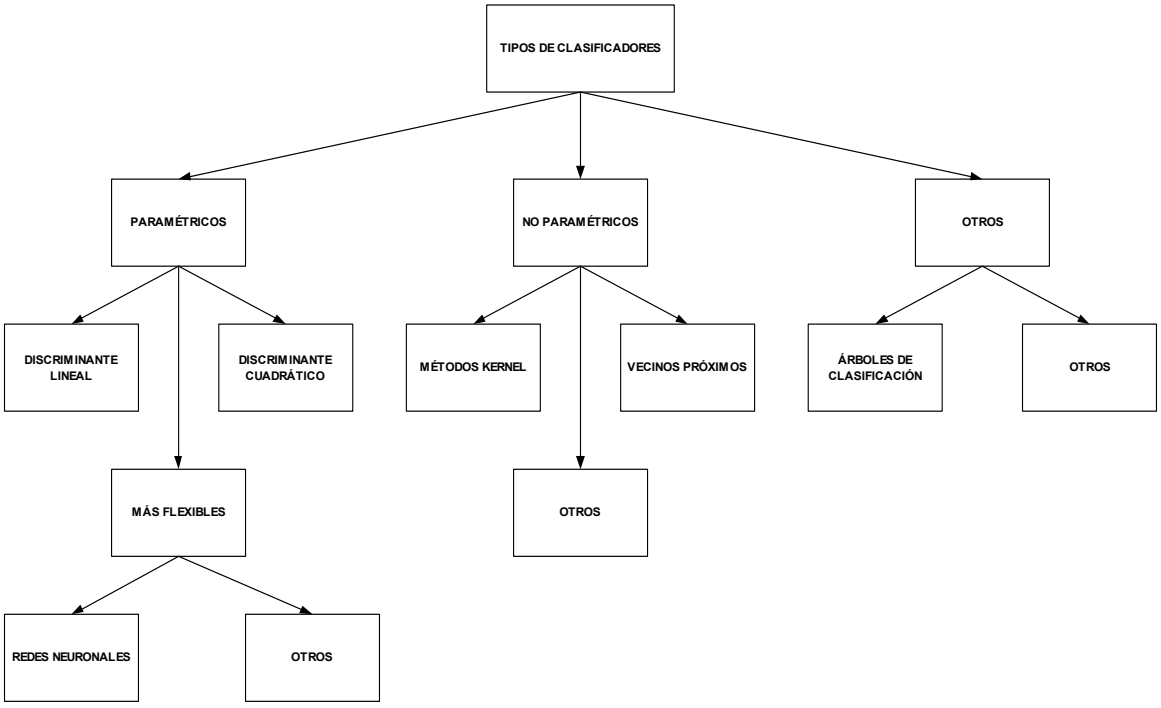

Figura 1.5. Alguno de los tipos de clasificadores más comunes

A continuación se hará una breve descripción de los diferentes tipos de clasificadores, insistiendo en aquellos que se han utilizado en este trabajo.

1.2.3.1 Clasificadores Paramétricos: Discriminantes Lineal y Cuadrático.

Consideraremos clasificadores paramétricos a aquellos cuyo diseño supone dar valores a una serie de parámetros. Aunque esta definición podría aplicarse a cualquier clasificador, dado que, siempre hay algún parámetro que ajustar, nos referimos, en concreto, a los casos en los que se asume una determinada estructura dependiente de cierto número de parámetros.

Otra cuestión es si esta estructura paramétrica se asume para las densidades de probabilidad de las clases, o para las probabilidades a posteriori. Ambas formas de

plantear el problema, se han mostrado efectivas en la práctica y habrá que optar por la que se crea más conveniente para un problema dado.

Los discriminantes lineal y cuadrático constituyen la opción más simple, pero también la menos flexible. A pesar de esta falta de flexibilidad, son muy usados en la práctica debido a la seria limitación que supone disponer de un número, habitualmente escaso, de patrones de entrenamiento. Se definen, como vimos, independientemente de la suposición de normalidad. En el caso particular del discriminante lineal, podemos escribir (para dos clases):

$$y = w^T x + w_0 \tag{1.11}$$

es decir, una combinación lineal de las componentes de x. Dependiendo de la dirección del vector w, obtendremos una mejor o peor separación de las clases. Se trata entonces de imponer criterios que permitan encontrar el valor óptimo de w. El umbral w_0 vendrá determinado por el criterio escogido.

Existen diferentes criterios para fijar w. El más clásico da lugar a lo que se conoce como discriminante de Fisher [Fisher 36], y consiste en maximizar la siguiente función:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{1.12}$$

donde S_W es la denominada matriz de dispersión intra-clases, definida como:

$$S_i = \sum_{x \in C_i} (x - \mu_i)(x - \mu_i)^T \tag{1.13}$$

$$S_W = S_1 + S_2 \tag{1.14}$$

,y S_B es la matriz de dispersión entre-clases, definida como:

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \tag{1.15}$$

Es fácil demostrar que el valor de w que maximiza la función anterior viene dado por:

$$w = S_W^{-1}(\mu_1 - \mu_2) \tag{1.16}$$

En definitiva, hemos pasado del problema original en múltiples dimensiones, a una sola dimensión. Esto puede resultar ventajoso, sobre todo cuando no se dispone de muchos patrones de entrenamiento. El valor umbral w_0 queda indeterminado y deberá fijarse con algún criterio razonable. Si las distribuciones de las clases son normales con matrices de covarianza iguales, obtendremos, como caso particular, el discriminante lineal de la sección anterior.

Así como para el discriminante lineal existen otros criterios que permiten fijar el valor de w [Fukunaga 90], para el discriminante cuadrático resulta mucho más complicado maximizar cualquier función de w, debido al gran numero de parámetros implicados. Por este motivo, se suele utilizar la expresión ya vista, sin más.

1.2.3.2 Redes Neuronales

Parece conveniente disponer de discriminantes paramétricos más flexibles que los anteriores. Entre las posibles opciones destacan por su gran popularidad las redes neuronales. Bajo este término se recogen una gran cantidad de sistemas computacionales que han experimentado un gran desarrollo en los últimos años. La característica común principal de estos sistemas es el estar inspirados en las redes neuronales naturales que constituyen los sistemas nerviosos de los seres vivos más evolucionados. Si bien, el origen biológico de estos sistemas carece de interés para la discusión que sigue.

Las redes neuronales implementan funciones complejas a base de interconectar elementos neuronales con funciones mucho más sencillas. La estructura de la neurona artificial depende en general de cada modelo, pero usualmente disponen de múltiples entradas con pesos asociados. Estos pesos son los parámetros a determinar, y representan la fuerza de las conexiones con otras neuronas. En los valores de los pesos queda almacenada la información del sistema. En las fases de entrenamiento o aprendizaje, estos pesos se hacen variar de una forma determinada para lograr el funcionamiento deseado.

De especial interés en el campo del reconocimiento de patrones resulta la red conocida como perceptrón, creado por Frank Rosenblatt [Rosenblatt 62], inicialmente

como modelo de retina artificial. Esencialmente es una red formada por una única capa, con múltiples entradas (cada una asociada a un elemento de visión, por ejemplo) y una salida binaria que indica si se ha detectado un patrón determinado a la entrada o no. En la figura 1.6 aparece la estructura del perceptrón.

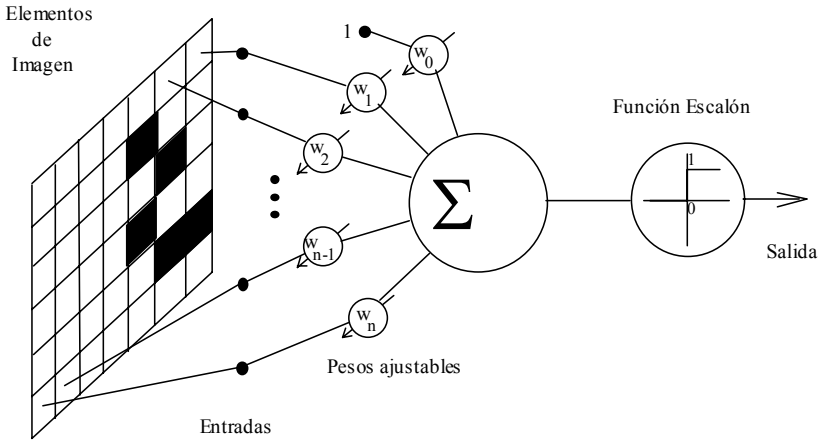

Figura 1.6. Esquema de la estructura del perceptrón.

El perceptrón implementa la función:

$$y = g\left(\sum_{i=0}^{n} w_i x_i\right) \tag{1.17}$$

siendo x_i las entradas y g la función de activación escalón o salto unitario. El peso w_0 se denomina sesgo o umbral, está conectado a una entrada 1 constante y tiene como misión aumentar la capacidad de representación del perceptrón, introduciendo una constante aditiva que da mayor flexibilidad, al poder controlar el umbral de disparo de la función escalón con independencia de las entradas actuales. Tenemos, en definitiva, otra forma de discriminante lineal, aunque obtenida por procedimientos diferentes a los planteados en la sección anterior.

Los problemas de capacidad de representación del perceptrón individual se resuelven al usar estructuras con múltiples capas en alimentación hacia delante. La estructura típica contiene varias capas conectadas totalmente (la salida de una neurona se propaga a todas las neuronas de la capa siguiente) como aparece en la figura 1.7.

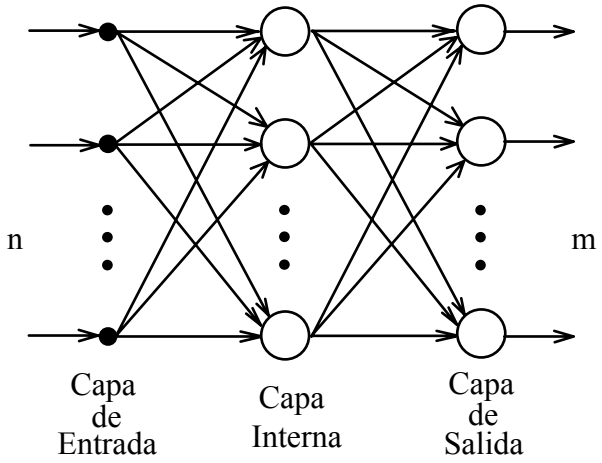

Figura 1.7. Estructura de un perceptrón multicapa con una capa interna

A esta estructura se le denomina perceptrón multicapa (MLP) con una capa interna. En general, se compone de una capa de entrada con n unidades, M unidades en la capa interna, y m unidades de salida. La relación entre las salidas y_k y las entradas x_i queda ahora de la siguiente manera:

$$y_k = \tilde{g}\left(\sum_{j=0}^{M} w_{kj}^{(2)} g\left(\sum_{i=0}^{n} w_{ji}^{(1)} x_i \right) \right) \tag{1.18}$$

donde $w_{ji}^{(1)}$ denota un peso en la capa de entrada, que va de la entrada i a la unidad j de la capa interna, y $w_{ji}^{(2)}$ representa un peso en la capa interna que va de la unidad j a la salida y_k. Se ha distinguido también entre la función de activación para la capa interna g y la función de activación para la capa de salida $\tilde{g}$.

La capa de entrada sólo ejerce la función de proporcionar el *fan-out* necesario para que se propaguen las entradas a la red a todos los elementos de la siguiente capa y por tanto no contiene unidades activas. La capa final de salida es la responsable de entregar los resultados finales. El resto de capas (internas) se denominan, tradicionalmente, ocultas, ya que no están en contacto directo con el exterior. Las unidades activas que componen la red son perceptrones con la modificación de tener a su salida, en lugar de la función salto, una función continua acotada como la sigmoide o la tangente hiperbólica:

$$\text{Función sigmoide: } g(\alpha) = \frac{1}{1+\exp(-\alpha)} \tag{1.19}$$

$$\text{Función tangente hiperbólica: } g(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}} \tag{1.20}$$

De esta manera la función del perceptrón es diferenciable, así como la red en conjunto, y se pueden emplear métodos de gradiente en su entrenamiento.

Una generalización de la función sigmoide que es usada con frecuencia como función de activación en la capa de salida es la función softmax [Bridle 90]:

$$y_k = \frac{\exp(\alpha_k)}{\sum_{k'=1}^{L} \exp(\alpha_{k'})} \tag{1.21}$$

Con esta función de activación, se normalizan las salidas de forma que estén en el rango (0,1) y sumen la unidad, lo cual es fundamental si éstas van a ser interpretadas como probabilidades.

El MLP de una sola capa interna con el número suficiente de unidades en ella es capaz de representar cualquier función continua con precisión arbitraria. Con dos capas internas, se supera la restricción de la continuidad. Contando con el número suficiente de unidades se aproxima, a la precisión requerida, cualquier función. La capacidad de representación no implica que esté resuelto el problema de determinar el conjunto de pesos adecuado para implementar la función deseada.

En las redes multicapas, el error se define usualmente como la suma de las diferencias al cuadrado entre las salidas que produce la red y las que debería producir, acumulada para cada patrón, debido a que ahora las salidas varían en un intervalo continuo. El objeto del entrenamiento es reducir dicho error para los patrones o parejas entrada/salida usados en el mismo, y de ello se hablará en más detalle en el capítulo 4.

1.2.3.3 Clasificadores No Paramétricos

En este tipo de clasificadores, no se asume ninguna forma paramétrica, ni de las densidades de probabilidad de las clases, ni de las probabilidades a posteriori. Los procedimientos que se siguen están precisamente orientados a la estimación de las densidades de probabilidad a partir de los patrones disponibles, o bien, a la estimación directa de las probabilidades a posteriori.

En el primer caso, tenemos los métodos que se denominan, genéricamente, de *kernel*, porque se basan precisamente en eso, en funciones *kernel* (por ejemplo, gaussianas), que de forma local, tratan de aproximar la forma de la densidad de probabilidad de la población de la que proceden los patrones [Hand 82].

Por otro lado, están los métodos que intentan estimar directamente las probabilidades a posteriori de las clases. Como ejemplo representativo de este tipo de clasificadores, podemos citar los de vecinos próximos [Dasarathy 91], que aproximan las probabilidades a posteriori para cada valor de x, asignándole la clase C_i más representada entre los k ejemplares de entrenamiento más cercanos.

Un resultado bastante significativo relacionado con los clasificadores de vecinos próximos es el que demuestra que el error asintótico (infinitos datos) es menor que dos veces el error de Bayes [Fukunaga 90]. Hay que tener en cuenta que este procedimiento de clasificación no usa ninguna información acerca de la estructura probabilística del problema.

En ambos casos, y especialmente en el primero, se necesitan muchos patrones para que estos métodos resulten eficientes, por lo cual, en ocasiones, su aplicabilidad práctica puede llegar a ser un tanto limitada.

1.2.3.4 Otros Clasificadores

Dentro de la categoría de otros clasificadores, incluimos aquellos que tratan del problema desde una perspectiva bastante diferente a los anteriores. Por ejemplo, métodos que consisten en particionar el espacio de características en regiones, y asignar una clase a cada región.

Dentro de esta filosofía, se encuentran los árboles de clasificación [Breiman 84], los cuales tienen la virtud de su fácil interpretabilidad, pero no resultan tan eficicientes como discriminadores.

CAPÍTULO 2. SOPORTE A LA IMPLEMENTACIÓN EN CLIPS DEL MODELO DE CONOCIMIENTO DE COMMONKADS

Las dificultades en el análisis, diseño e implementación de un Sistema Basado en el Conocimiento son enormes, mucho mayores que las de cualquier sistema software convencional. El primer escollo está en el proceso de adquisición de conocimiento, que trata de recoger y formalizar ('representar') la información disponible, para ser manipulada adecuadamente. En lo que se refiere al diseño, tradicionalmente, ha descansado directamente sobre mecanismos particulares de implementación, tales como redes semánticas, reglas, etc. Esto implica que las estructuras del razonamiento y el conocimiento sobre el que se aplican, están indisolublemente unidos. Como consecuencia, se hace imposible la reutilización de las estrategias de inferencia en otros dominios distintos de aquel para el que se han creado inicialmente. Con el fin de solucionar estas dificultades y simplificar el problema de la adquisición de conocimiento, surgen en la última década diversas metodologías, como ya se explicó en el capítulo anterior.

Estas metodologías distinguen una serie de modelos de diferentes aspectos del conocimiento, y en particular, se describieron brevemente los modelos que forman la metodología CommonKADS. De entre ellos, destacan el Modelo de Conocimiento y el Modelo de Diseño, de los cuales nos ocupamos extensamente en este capítulo. Así, dentro del Modelo de Diseño, se propone una arquitectura software basada en el lenguaje CLIPS que sirve como soporte a la implementación del Modelo de Conocimiento de CommonKADS. El proceso se completa con la adición de una herramienta que automatiza parcialmente la traducción de los objetos CommonKADS a objetos CLIPS.

2.1 Descripción de los Modelos de Conocimiento y de Diseño de CommonKADS

En esta sección se describirán los dos modelos de la metodología CommonKADS a los cuales se ha dedicado una atención especial. En primer lugar, se muestran los diferentes tipos de conocimiento que se consideran en el Modelo de Conocimiento y posteriormente se introduce el Modelo de Diseño, resaltando el principio de conservación de la estructura como fundamental para la transparencia y mantenibilidad del sistema.

2.1.1 Modelo de Conocimiento de CommonKADS

Como ya se comentó en la parte de fundamentos, el Modelo de Conocimiento es el más importante de cuantos se definen en esta metodología. Proporciona una especificación de los datos y estructuras de conocimiento requeridos en una aplicación. Esta especificación viene dada en el vocabulario propio de la aplicación, dejando cualquier detalle de implementación para la fase de diseño. El ejemplo más claro de esta separación de los aspectos de análisis y de diseño, lo constituye el término “regla”. En el Modelo de Conocimiento nos referiremos a este término en el sentido en el que lo usa un humano, de forma natural. Si finalmente, estas reglas “naturales” son implementadas a través de un formalismo de reglas o no, es una cuestión aparte, puramente de diseño y no relevante durante el análisis.

El Modelo de Conocimiento tiene una estructura que es esencialmente similar a los modelos de análisis tradicionales en Ingeniería del Software. Así, la tarea de

razonamiento se describe a través de una descomposición jerárquica en funciones o procesos. Los datos y tipos de conocimiento sobre los que operan estas funciones se describen a través de un esquema que recuerda a un modelo de datos o un modelo de objetos. Las notaciones son también similares a las encontradas en otro tipo de modelos, por ejemplo, la notación usada en UML (Unified Modelling Language) [Booch 98]. A pesar de todas estas similitudes, también hay diferencias que se expondrán en secciones posteriores.

En el Modelo de Conocimiento podemos distinguir tres categorías.

La primera categoría es el llamado Conocimiento de Dominio, y especifica el conocimiento específico de un dominio y los tipos de información concernientes a una aplicación particular.

La segunda categoría es el Conocimiento de Inferencia, y describe los pasos básicos de razonamiento que se pretenden llevar a cabo, usando el Conocimiento de Dominio.

Por último, la tercera categoría es el Conocimiento de Tarea, y describe los objetivos que persigue una aplicación y como alcanzar estos objetivos a través de una descomposición en subtareas e inferencias. En la figura 2.1 se muestran esquematizadas estas tres categorías de conocimiento.

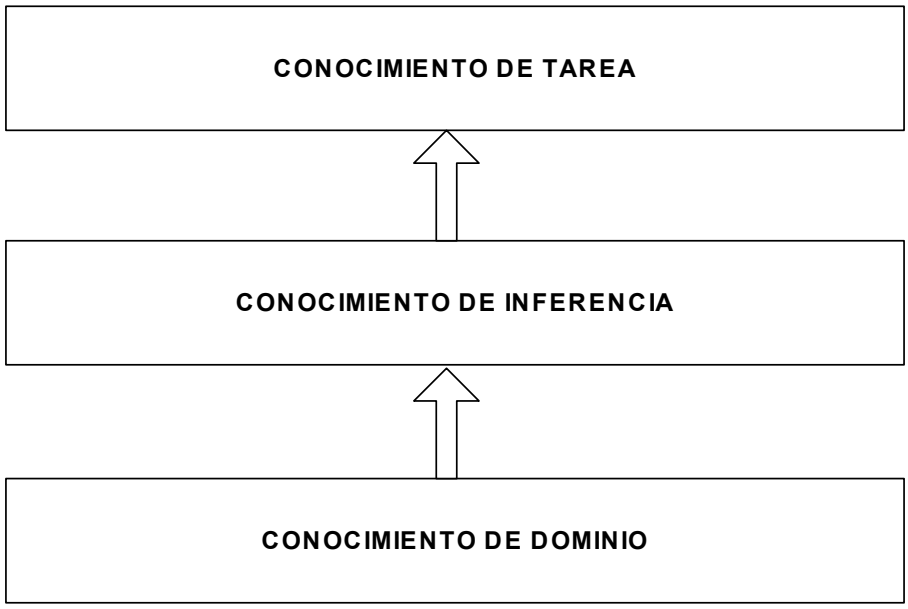

Figura 2.1. Categorías que se distinguen en el Modelo de Conocimiento

En secciones posteriores, se estudiarán, en detalle, estas tres categorías. Para describir las construcciones que se definen en cada una de ellas, se utilizarán gráficos y texto. La descripción textual está basada en el lenguaje pseudoformal CML (*CommonKADS Modelling Language)*, creado para la especificación de Modelos de Conocimiento en CommonKADS.

2.1.1.1 Conocimiento de Dominio

El Conocimiento de Dominio describe el conocimiento estático específico del dominio de una aplicación. Esta descripción se realiza a dos niveles diferentes:

- Esquema de Dominio: constituye una descripción esquemática de las clases de objetos o entidades presentes, y sus relaciones. En Ingeniería del Software, sería el equivalente a un modelo de datos o modelo de objetos.
- Base de Conocimiento: contiene instancias de los tipos de conocimiento especificados en el Esquema de Dominio.

Especificación de un Esquema de Dominio

El Modelo de Conocimiento proporciona una serie de construcciones que permiten especificar el Esquema de Dominio de una aplicación. La mayoría de estas construcciones son similares a las encontradas en modelos de datos orientados a objetos. Se distinguen tres construcciones básicas que son CONCEPT**,** RELATION**,** y RULE-TYPE, y se incluyen también otras adicionales como SUB-TYPE-OF, relacionada con las anteriores.

Para ilustrar este tipo de construcciones, se utilizarán ejemplos de un problema simple de diagnóstico de averías en coches.

Concepto (CONCEPT***)***

Un Concepto describe un conjunto de objetos o instancias del dominio, que comparten características similares. Viene a ser lo mismo que la noción de Clase en otro tipo de modelos, pero a diferencia de las aproximaciones orientadas a objetos, no se incluyen funciones o métodos en la definición del mismo.

Las características de un Concepto se especifican a través de Atributos (ATTRIBUTE). Un Atributo contiene un valor, una pieza de información que puede aparecer en instancias del Concepto. Los tipos de valores que puede contener un Atributo son variados. Algunos de los más comunes son números naturales (natural), números reales (real), "booleanos" (boolean), o cadenas de caracteres (string). Se entiende además que estas piezas de información son atómicas, es decir, que no pueden ser, por ejemplo, instancias de otro Concepto.

Gráficamente, podemos representar un Concepto como se muestra en el ejemplo de la figura 2.2.

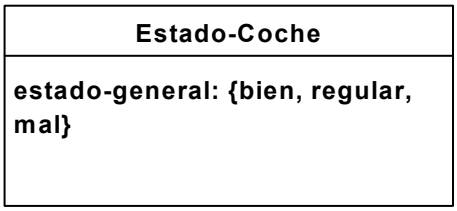

Figura 2.2. Concepto Estado-Coche con Atributo estado-general

En CML vendría expresado de la siguiente manera:

```
CONCEPT Estado-Coche;
   ATTRIBUTES:
      estado-general: {bien, regular, mal};
END CONCEPT Estado-Coche;
```

Relación (RELATION***)***

Las Relaciones entre Conceptos se definen con la construcción RELATION. Necesitan de la especificación de Argumentos (ARGUMENTS), cada uno de ellos con una Cardinalidad (CARDINALITY) asociada, que indica su participación en la Relación. Además puede especificarse un Rol (ROLE), que identifica el papel que el argumento juega en la Relación.

Las Relaciones pueden tener cualquier número de Argumentos, aunque la gran mayoría tienen sólo dos. Por este motivo, se define la construcción BINARY-RELATION, como un caso especial de la anterior. Al igual que los Conceptos, las Relaciones también pueden tener Atributos, que son valores que dependen del contexto de la Relación. En la figura 2.3 se muestra un ejemplo gráfico simple de Relación entre los Conceptos Coche y Persona.

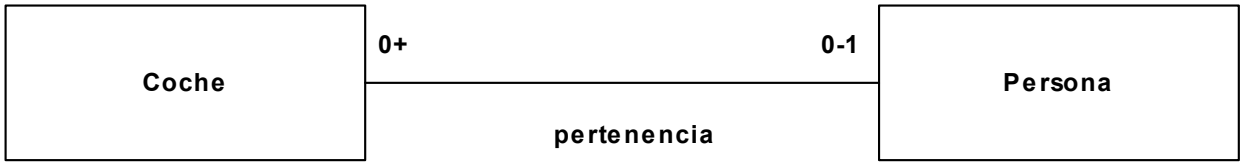

Figura 2.3 Relación de pertenencia entre el Concepto Coche y el Concepto Persona

En CML:

```
CONCEPT Coche;
END CONCEPT Coche;

CONCEPT Persona;
END CONCEPT Persona;

BINARY-RELATION pertenecer-a;
   INVERSE: poseer;
   ARGUMENT-1: Coche;
      CARDINALITY: 0+;
   ARGUMENT-2: Persona;
      CARDINALITY: 0-1;
END BINARY-RELATION pertenecer-a;
```

Subtipo de (SUB-TYPE-OF***)***

El Modelo de Conocimiento soporta la especificación de relaciones de generalización/especialización. Así, los Conceptos pueden ser organizados en jerarquías de Subtipos, a través de la construcción SUB-TYPE-OF. Las Relaciones también pueden tener Subtipos. Esta jerarquización supone que Atributos de los Conceptos más generales son heredados por Conceptos más particulares en la jerarquía. Por otro lado, se introduce una especialización que suele consistir en:

- Añadir un nuevo Atributo, o una nueva participación en una Relación.
- Restringir el conjunto de valores de un Atributo, o el tipo de estos.
- Restringir la cardinalidad en una Relación.

  En la figura 2.4 se muestra un ejemplo de relaciones Subtipo de.

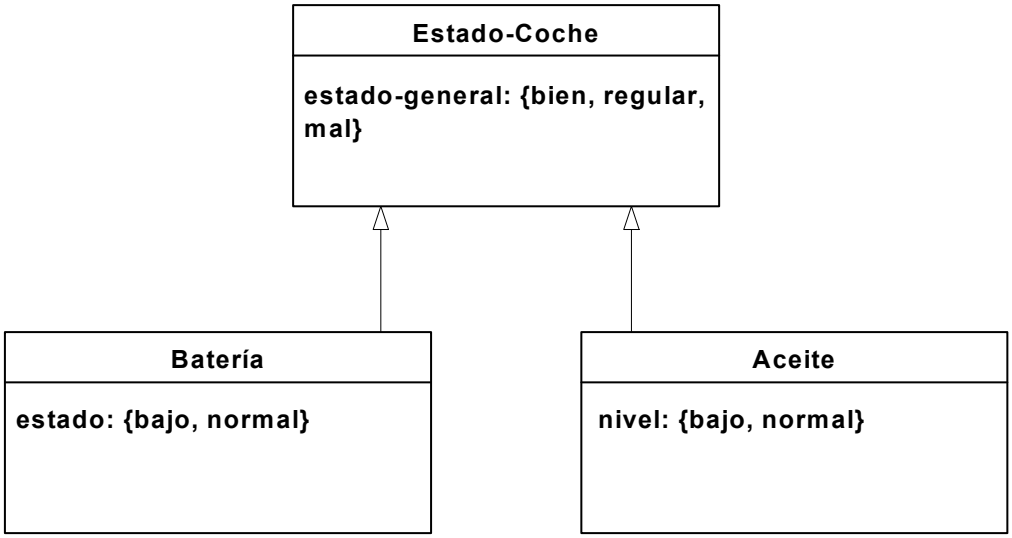

Figura 2.4. Ejemplos de Relaciones Subtipo de

En CML:

```
CONCEPT Batería;
    SUB-TYPE-OF Estado-Coche;
    ATTRIBUTES:
        estado: {bajo, normal};
END CONCEPT Batería;

CONCEPT Aceite;
    SUB-TYPE-OF Estado-Coche;
    ATTRIBUTES:
        nivel: {bajo, normal};
END CONCEPT Aceite;
```

Tipo de Regla (RULE-TYPE***)***

Hasta ahora, las construcciones descritas son muy similares a las encontradas en otros modelos de datos tradicionales. Sin embargo, en el modelado de conocimiento se precisan construcciones que den cuenta de un tipo particular de relación, como la que se muestra en los siguientes ejemplos de dependencias entre Conceptos:

Batería.estado = bajo **or** Aceite.nivel = bajo ⇒ Motor.estado = no-arranca;

Tanque-Gasolina.nivel = vacío ⇒ Motor.estado = se-para;

Tenemos entonces dependencias lógicas entre valores de Atributos de Conceptos. Este tipo de relaciones son de un carácter especial y se modelan a través de la construcción RULE-TYPE. De esta manera, el Tipo de Regla para modelar las dependencias anteriores, sería de la siguiente manera:

```
RULE-TYPE Dependencia-Estados;
   ANTECEDENT: Estado-Coche;
   CONSEQUENT: Estado-Coche;
   CONNECTION-SYMBOL: causa;
END RULE-TYPE Dependencia-Estados;
```

Las Expresiones (EXPRESSIONS) anteriores vienen a ser, entonces, instancias de este tipo de construcción. El Símbolo Conector (CONNECTION-SYMBOL) relaciona el Antecedente (ANTECEDENT) con el consecuente (CONSEQUENT), y su nombre debería reflejar el tipo de relación entre ambos.

En la figura 2.5 se muestra la representación gráfica del Tipo de Regla.

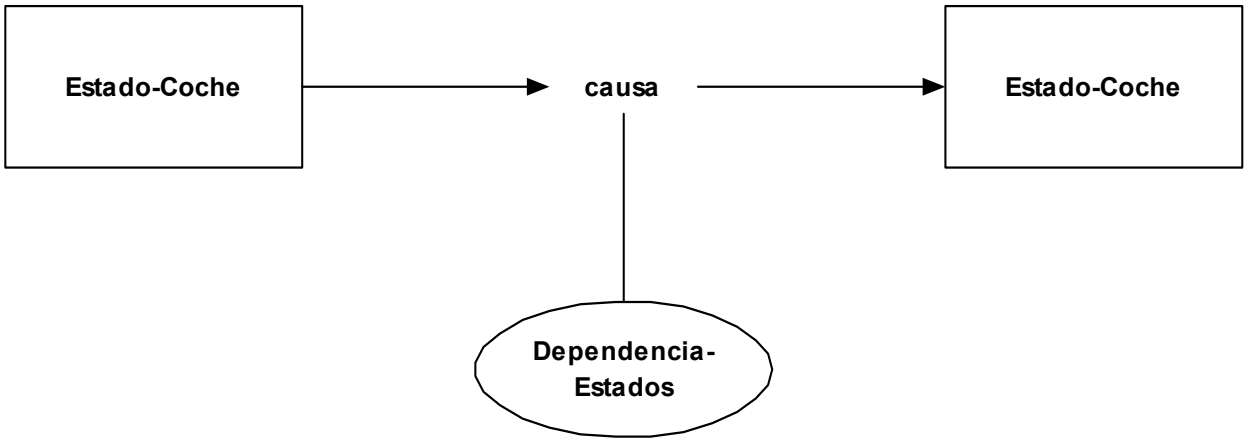

Figura 2.5. Representación Gráfica de la construcción Tipo de Regla

Es importante reseñar que con la construcción Tipo de Regla, podemos estructurar una base de conocimiento en grupos de reglas que comparten una estructura similar.

Bases de Conocimiento

Así como un Esquema de Dominio describe la estructura del conocimiento del dominio, una Base de Conocimiento contiene instancias de los tipos de conocimiento. Como ejemplo de Base de Conocimiento mostramos la siguiente:

```
KNOWLEDGE-BASE Relaciones-Causa-Efecto-Coche;
   USES:
      Dependencia-Estados FROM Esquema-Diagnóstico-Paciente;
   EXPRESSIONS:
      Batería.estado = bajo OR Aceite.nivel = bajo CAUSA Motor.estado = no-arranca;
      Tanque-Gasolina.nivel = vacío CAUSA Motor-estado = se-para;
END KNOWLEDGE-BASE Relaciones-Causa-Efecto-Coche;
```

En su definición se indica el Tipo de Regla usado y el Esquema-de-Dominio donde está definido.

La separación de Esquema-de-Dominio y Base de Conocimiento supone un replanteamiento del término adquisición del conocimiento, ya que sugiere los siguientes pasos a seguir:

- Definir los tipos de conocimiento presentes.
- Instanciar estos tipos de conocimiento para formar Bases de Conocimiento.

2.1.1.2 Conocimiento de Inferencia

En el Conocimiento de Dominio se describe como las estructuras estáticas descritas en la sección anterior, pueden ser usadas para llevar a cabo los procesos de razonamiento necesarios. Los componentes principales del Conocimiento de Inferencia son las Inferencias, los Roles de Conocimiento y las Funciones de Transferencia.

Inferencia y Rol de Conocimiento (INFERENCE, KNOWLEDGE-ROLE)

El Conocimiento de Inferencia en el Modelo de Conocimiento especifica el nivel más bajo de descomposición funcional. A las unidades de razonamiento más básicas se les denomina Inferencias y son modeladas a través de la construcción INFERENCE.

Es importante tener en cuenta que una Inferencia queda totalmente descrita por una especificación declarativa de su entrada y salida. El proceso interno de la Inferencia es, a estos efectos, una "caja negra", carente de interés para el modelado de conocimiento. En cierto modo, esto constituye una guía para decidir que se considera como Inferencia a la hora de modelar el proceso de razonamiento en un problema determinado.

Normalmente, una Inferencia usará conocimiento contenido en alguna Base de Conocimiento, para llevar a cabo su cometido, pero su relación con el Conocimiento de Dominio no es directa sino a través de los llamados Roles de Conocimiento. Esto facilita la reutilización de Inferencias en otros dominios.

Los Roles de Conocimiento, modelados por la construcción KNOWLEDGE-ROLE, vienen descritos por nombres generales y abstractos que indican su papel o *rol* en el proceso de razonamiento. Hay que distinguir dos tipos de Roles, por un lado los Roles Dinámicos (DYNAMIC), y por otro, los Roles Estáticos (STATIC). Ambos se vinculan con el dominio a través de su especificación DOMAIN-MAPPING.

Los Roles Dinámicos representan las entradas y salidas que cambian en cada invocación de la Inferencia. Los Roles Estáticos son estables en el tiempo y especifican el conocimiento del dominio utilizado para llevar a cabo la Inferencia.

A continuación, se muestra un ejemplo de lo expuesto:

```
KNOWLEDGE-ROLE Fallo-Observado;
   TYPE: DYNAMIC;
   DOMAIN-MAPPING: Estado-Coche;
END KNOWLEDGE-ROLE Fallo-Observado;

KNOWLEDGE-ROLE Hipótesis;
   TYPE: DYNAMIC;
   DOMAIN-MAPPING: Estado-Coche;
END KNOWLEDGE-ROLE Hipótesis;

KNOWLEDGE-ROLE Modelo-Causal;
```

```
    TYPE: STATIC;
    DOMAIN-MAPPING: Dependencia-Estados FROM Relaciones-Causa-Efecto-Coche;
END KNOWLEDGE-ROLE Modelo-Causal;

INFERENCE Buscar-Causa;
    ROLES:
    INPUT: Fallo-Observado;
    OUTPUT: Hipótesis;
    STATIC: Modelo-Causal;
    SPECIFICATION:
    "Cada vez que se invoca esta inferencia, genera una posible hipótesis acerca de la
    causa que pudo haber provocado los síntomas";
END INFERENCE Buscar-Causa;
```

En la figura 2.6, se representa gráficamente la Inferencia Buscar-Causa y su relación con el dominio a través de los Roles de Conocimiento.

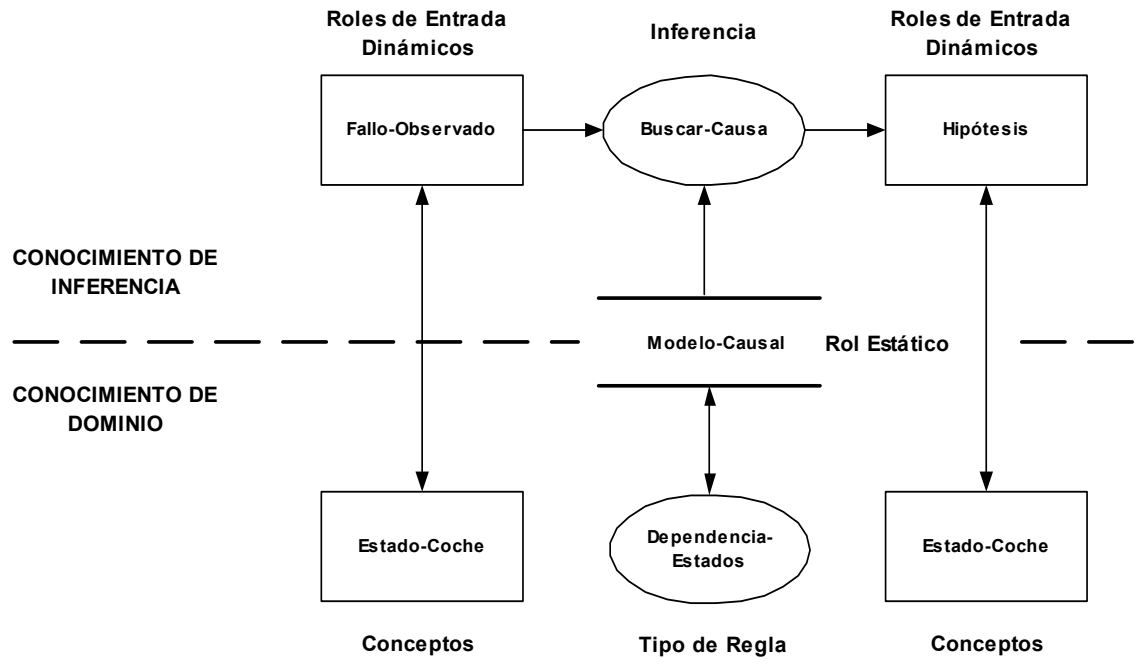

Figura 2.6 Inferencia Buscar-Causa y su relación con el dominio

Función de Transferencia (TRANSFER-FUNCTION)

En el modelo de conocimiento nos abstraemos de la comunicación con otros agentes (usuarios y otros sistemas), haciendo énfasis en el proceso de razonamiento. Sin

embargo, no es posible olvidar por completo la interacción externa. De hecho, algunas de estas interacciones juegan un determinado papel en el proceso de razonamiento en si mismo, por ejemplo, obteniendo información adicional acerca del problema en cuestión. Por este motivo, se introduce la noción de Función de Transferencia.

Una Función de Transferencia es una función que transfiere información entre el agente de razonamiento, descrito en el Modelo de Conocimiento, y otros sistemas o usuarios. Al igual que ocurre con las Inferencias, se consideran "cajas negras" desde el punto de vista del modelado de conocimiento.

Las Funciones de Transferencia tienen nombres estándar que responden a dos cuestiones fundamentales: ¿quién tiene la iniciativa en la comunicación? y ¿quién está en posesión de la pieza de información a transferir?. Basándose en estas cuestiones, se distinguen cuatro Funciones de Transferencia:

- Obtener: el agente de razonamiento solicita información de un agente externo. El agente de razonamiento toma la iniciativa y el agente externo tiene la información.
- Recibir: el agente de razonamiento recibe información del agente externo. El agente externo toma la iniciativa y también tiene la información.
- Presentar: el agente de razonamiento presenta la información al agente externo. El agente de razonamiento tiene la información y toma la iniciativa.
- Proporcionar: el sistema le proporciona información a un agente externo a petición de éste. El agente externo toma la iniciativa y el agente de razonamiento tiene la información.

En la figura 2.7, se representan esquemáticamente estas cuatro situaciones.

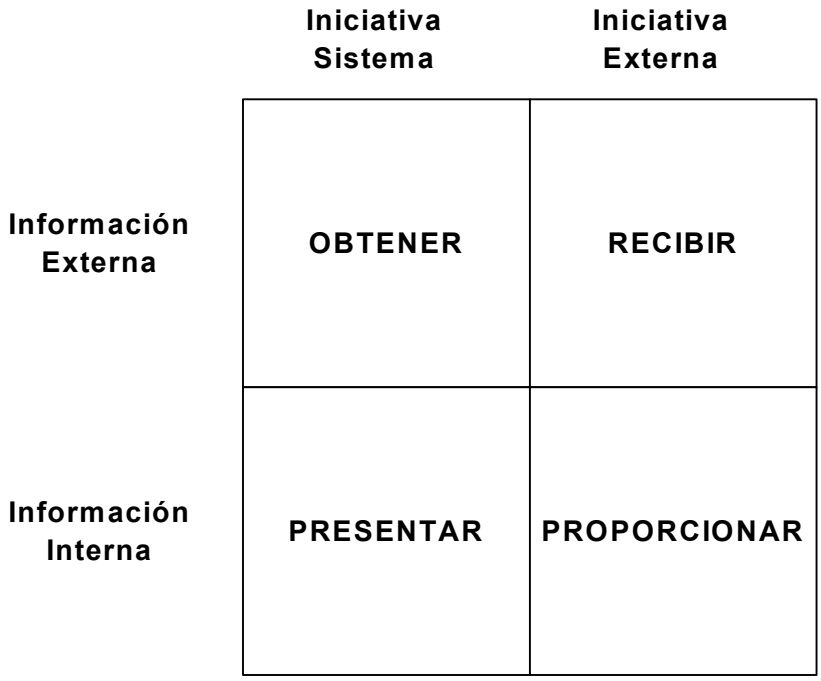

Figura 2.7. Tipología de las Funciones de Transferencia

2.1.1.3 Conocimiento de Tarea

Un aspecto fundamental del conocimiento es lo qué pretendemos hacer con él, en definitiva, qué objetivos pretendemos alcanzar aplicando conocimiento. El Conocimiento de Tarea es la categoría que da respuesta a estos interrogantes. Generalmente se describe de una forma jerárquica por descomposición en tareas cada vez más simples, hasta llegar al nivel de las Inferencias y Funciones de Transferencia.

En el Conocimiento de Tarea se distinguen dos tipos de conocimiento principales: la Tarea y el Método de Tarea. Una Tarea define un objetivo de razonamiento en términos de un par entrada-salida. Un Método de Tarea describe como puede realizarse la Tarea a través de una descomposición en subfunciones, más una estructura de control sobre estas subfunciones.

Tarea (TASK**)**

Para el ejemplo del diagnóstico de averías en un coche, podemos escribir la Tarea Diagnóstico-Averías-Coche de la siguiente manera:

```
TASK Diagnóstico-Averías-Coche;
   GOAL:
      "Encontrar una causa probable para la avería";
   ROLES:
      INPUT: Fallo-Observado: "Fallo observado en el comportamiento del coche";
      OUTPUT:  Causa-Probable: "hipótesis corroborada por la evidencia";
                  Evidencia: "Conjunto de hechos observados durante el proceso de
               diagnóstico";
   SPECIFICATION:
      "Encontrar la causa que explique la avería y sea consistente con la evidencia
   obtenida";
END TASK Diagnóstico-Averías-Coche;
```

Como se puede observar, su especificación es similar a la de una Inferencia, pero se diferencian en dos cosas fundamentales:

- No se incluyen Roles Estáticos en la definición.
- No se especifica directamente la relación con el dominio. Se hace de forma indirecta a través de los Roles en la estructura de control.

En la figura 2.8 se muestra la tarea de diagnóstico y su descomposición en inferencias.

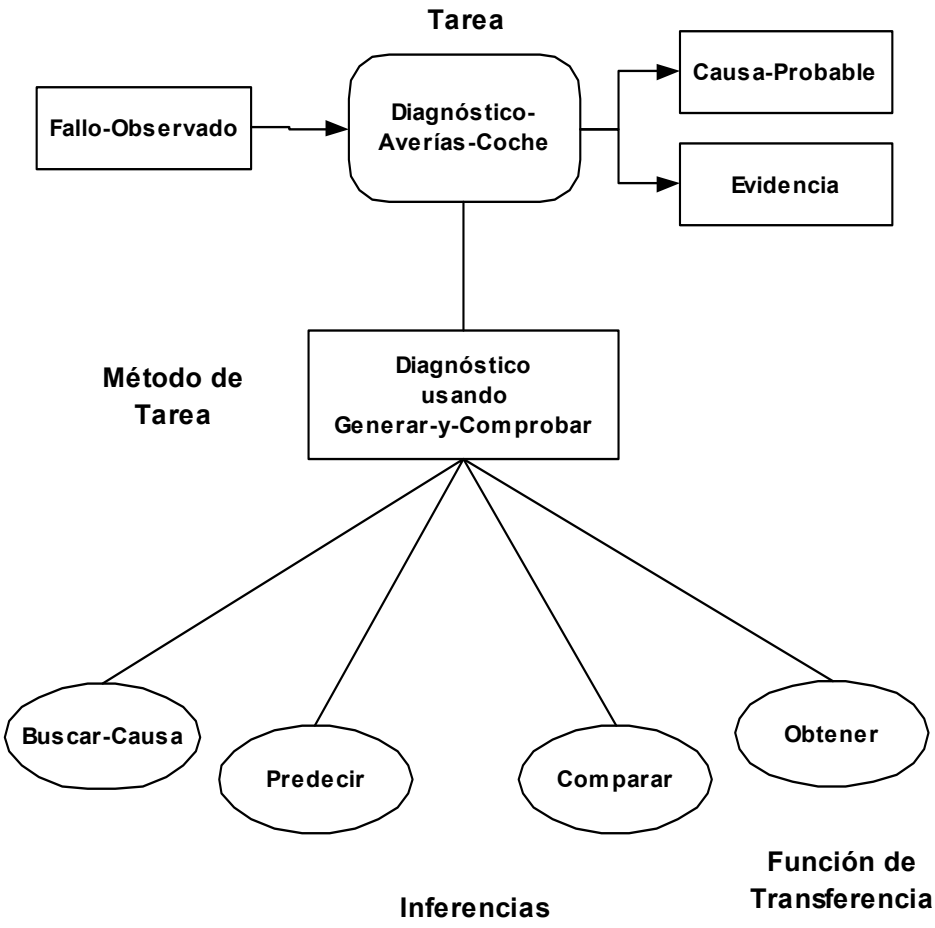

Figura 2.8. Tarea de diagnóstico y su descomposición en Inferencias

Método de Tarea (TASK-METHOD)

El núcleo del Método de Tarea lo forma la estructura de control, que describe en qué orden se llevan a cabo las subfunciones (subtareas e inferencias). La estructura de control se lee como si fuera un pequeño programa en el que las subfunciones son los procedimientos y los Roles actúan como parámetros de estas subfunciones. En definitiva, la estructura de control pretende capturar la estrategia de razonamiento procedural empleada en la resolución de un problema.

A continuación se muestra en CML, el Método de Tarea del problema de diagnóstico de averías de coches:

```
TASK-METHOD Diagnóstico-usando-Generar-y-Comprobar;
   REALIZES: Diagnóstico-Averías-Coche;
   DECOMPOSITION:
      INFERENCES: Buscar-Causa, Predecir, Comparar;
      TRANSFER-FUNCTIONS: Obtener;
   ROLES:
      INTERMEDIATE:
         Hipótesis: "Hipótesis inicial sobre la causa de la avería";
         Evidencia-Esperada: "Evidencia que cabría esperar si la hipótesis fuera
      correcta";
         Evidencia-Encontrada: "Evidencia real encontrada";
   CONTROL-STRUCTURE:
      REPEAT
         Buscar-Causa (Fallo-Observado → Hipótesis);
         Predecir (Hipótesis → Evidencia-Esperada);
         Obtener (Evidencia-Esperada →Evidencia-Encontrada);
         Evidencia := Evidencia ADD Evidencia-Encontrada;
         Compare (Evidencia-Encontrada + Evidencia-Esperada → Resultado);
      UNTIL
         "Resultado = verdad o no hayan más hipótesis";
      END REPEAT
      IF Resultado == verdad
         THEN Causa-Probable := Hipótesis;
         ELSE "No se encuentra solución";
      END IF
END TASK-METHOD Diagnóstico-usando-Generar-y-Comprobar;
```

2.1.2 Modelo de Diseño de CommonKADS

En la sección anterior dejamos de lado, intencionadamente, todas aquellas cuestiones que tenían que ver con la implementación, por no considerarse de interés en el proceso de modelado de conocimiento. En cualquier caso, resulta evidente la necesidad de disponer de algún procedimiento que permita convertir el Modelo de Conocimiento en un sistema software implementable. El Modelo de Diseño responde a esta demanda, ya que describe la estructura del sistema software necesario, en términos de los subsistemas, módulos, mecanismos computacionales y construcciones requeridas en la implementación. De esta manera, el Modelo de Conocimiento constituye la entrada

principal al proceso de diseño, especificando los requisitos que serán trasladados a especificaciones software.

Un elemento fundamental del Modelo de Diseño es la arquitectura software. Ésta describe la estructura del sistema software en términos de subsistemas y módulos, así como el régimen de control, a través del cual interactúan estos subsistemas. En este trabajo se ha tomado como referencia la arquitectura software propuesta en [Schreiber 99].

2.1.2.1 Preservación de la Estructura en el Proceso de Diseño

En principio sería posible pasar del Modelo de Conocimiento a una arquitectura software basada exclusivamente en reglas de producción. Sin embargo, un diseño de este tipo impediría la distinción entre las diferentes categorías de conocimiento establecidas en el modelo, quedando mezcladas en la base de reglas. Desde un punto de vista metodológico, deberíamos tender a un diseño que preserve la información y las estructuras definidas en el Modelo de Conocimiento. De esta manera, conseguiremos que el sistema software resultante satisfaga ciertos criterios de calidad, tales como:

- Reusabilidad del código: con la conservación de la estructura, se hacen explícitos el propósito y el papel que juegan los distintos fragmentos de código, permitiendo su reutilización en otras aplicaciones. Cuando hablamos de fragmentos de código, nos referimos tanto a la implementación de inferencias, como a implementaciones más complejas de tareas compuestas por varias inferencias.
- Mantenibilidad y adaptabilidad: el mantenimiento del sistema se simplifica notablemente, ya que la estructura, presente también en la implementación, permite identificar más fácilmente posibles fuentes de errores o inconsistencias, y relacionarlos con una parte específica del modelo. Además, resulta más sencillo añadir mejoras con el fin de aumentar su funcionalidad.
- Poder explicativo: la necesidad de explicar por qué se ha seguido un determinado proceso de razonamiento es una característica típica de los Sistemas Basados en el Conocimiento. Con la preservación de la estructura, se posibilita la explicación en el vocabulario propio del Modelo de Conocimiento, resultando factible contestar a preguntas del tipo:

- ¿En qué paso elemental de la resolución del problema se usa una determinada pieza de conocimiento y qué papel juega en la inferencia que lo utiliza?.
- ¿Cuando y por qué es usado para resolver un problema particular?.

Hay que indicar, por último, que la preservación de la estructura en el diseño se usa también en Ingeniería del Software, especialmente en el área de modelado y diseño orientado a objetos, con motivaciones similares a las que acabamos de comentar.

2.2 Implementación del Modelo de Diseño

En esta sección, con las guías generales de la sección 2.1, se propone un Modelo de Diseño concreto susceptible de ser utilizado con cualquier dominio. Adicionalmente, se plantea la automatización del proceso de creación de Modelos de Diseño con la ayuda de herramientas de construcción de compiladores.

2.2.1 Descripción de la Arquitectura Software Utilizada

En la figura 2.9 se muestran los elementos fundamentales que componen la arquitectura software diseñada para la implementación.

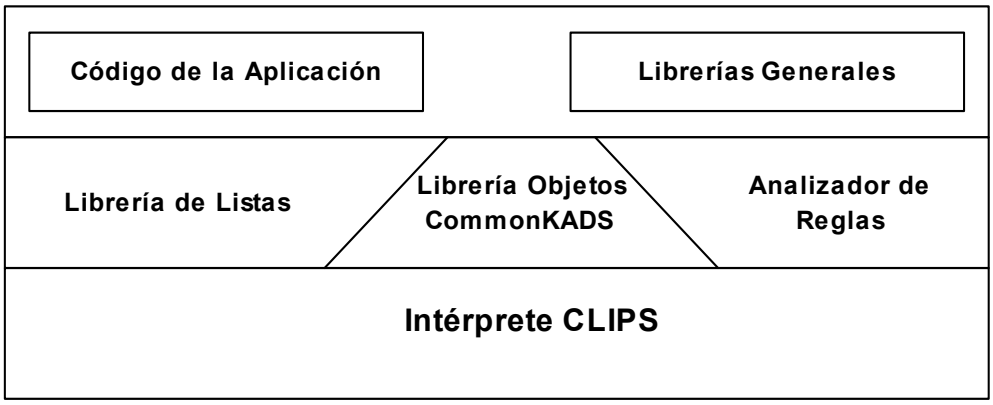

Figura 2.9. Arquitectura software de implementación

La plataforma de implementación elegida ha sido el lenguaje CLIPS, herramienta de diseño para Sistemas Basados en el Conocimiento. Por encima del nivel de intérprete, encontramos una capa formada por una serie de librerías CLIPS. Estas librerías implementan, de forma genérica, el *mapping* de las estructuras y tipos de conocimiento descritos en el Modelo de Conocimiento, a objetos y funciones de CLIPS. Finalmente,

en la última capa se sitúa el código de la aplicación particular considerada, que hará uso de estas librerías.

Para hacer uso de la arquitectura propuesta, el implementador del SBC deberá, a partir del Modelo de Conocimiento descrito en CML, instanciar las clases CLIPS generales definidas en la librería de objetos de CommonKADS para generar los objetos CLIPS particulares de una aplicación (conceptos, inferencias, tareas,...). Además quedarían por completar los cuerpos y funciones de las inferencias y tareas.

Con todo esto, la ejecución del SBC se iniciará con un mensaje de ejecución enviado a la tarea de más alto nivel.

En los apartados que siguen, se describe en detalle cada una de las capas que aparecen en la figura.

2.2.1.1 Plataforma de Implementación: CLIPS

El CLIPS (*C Language Integrated Production System*) es lenguaje pensado para el diseño de Sistemas Basados en el Conocimiento [Giarratano 98]. Fue creado en 1984 en el Centro Espacial Johnson de la NASA. Entre sus numerosas ventajas cabe citar su gratuidad, el hecho de ser multiplataforma y de código abierto, interacciona fácilmente con otros lenguajes y posee un motor de inferencia eficiente. Se ha utilizado la versión más reciente 6.1.

Hay tres formas de representar el conocimiento en CLIPS:

- A través de reglas de producción, usadas para representar conocimiento de tipo heurístico.
- A través de funciones, usadas para representar conocimiento procedural.
- A través de objetos, soportándose las características generales de la programación orientada a objetos.

Ha sido esta flexibilidad a la hora de representar el conocimiento, la razón principal por la cual hemos elegido este lenguaje como plataforma de implementación.

2.2.1.2 Librería principal: Implementación en CLIPS de los Objetos Genéricos del Modelo de Conocimiento

De las librerías creadas, la principal es la que contiene el código CLIPS necesario para la implementación de las estructuras de conocimiento descritas en la sección 2.1 (ver en el apéndice el listado completo de esta librería llamada ***CommonKADS.clp***). Para la creación de esta librería, se ha hecho uso, fundamentalmente, de la representación orientada a objetos.

Para cada objeto del CommonKADS se mostrará primero su sintaxis CML, a continuación su equivalente en CLIPS, y finalmente un ejemplo ilustrativo, para lo cual se volverán a usar los que aparecen en la descripción del Modelo de Conocimiento. Hay que aclarar que no se ha utilizado el lenguaje CML, estrictamente, sino una versión reducida con ciertas variantes de conveniencia (la sintaxis completa se encuentra en el apéndice). De todas formas, las construcciones principales están igualmente contempladas, e incluso se ha añadido una mayor flexibilidad y versatilidad a algunas de ellas.

En la figura 2.10, se muestran las clases CLIPS definidas en esta librería, y en lo que sigue, se procederá a explicar cada una de ellas.

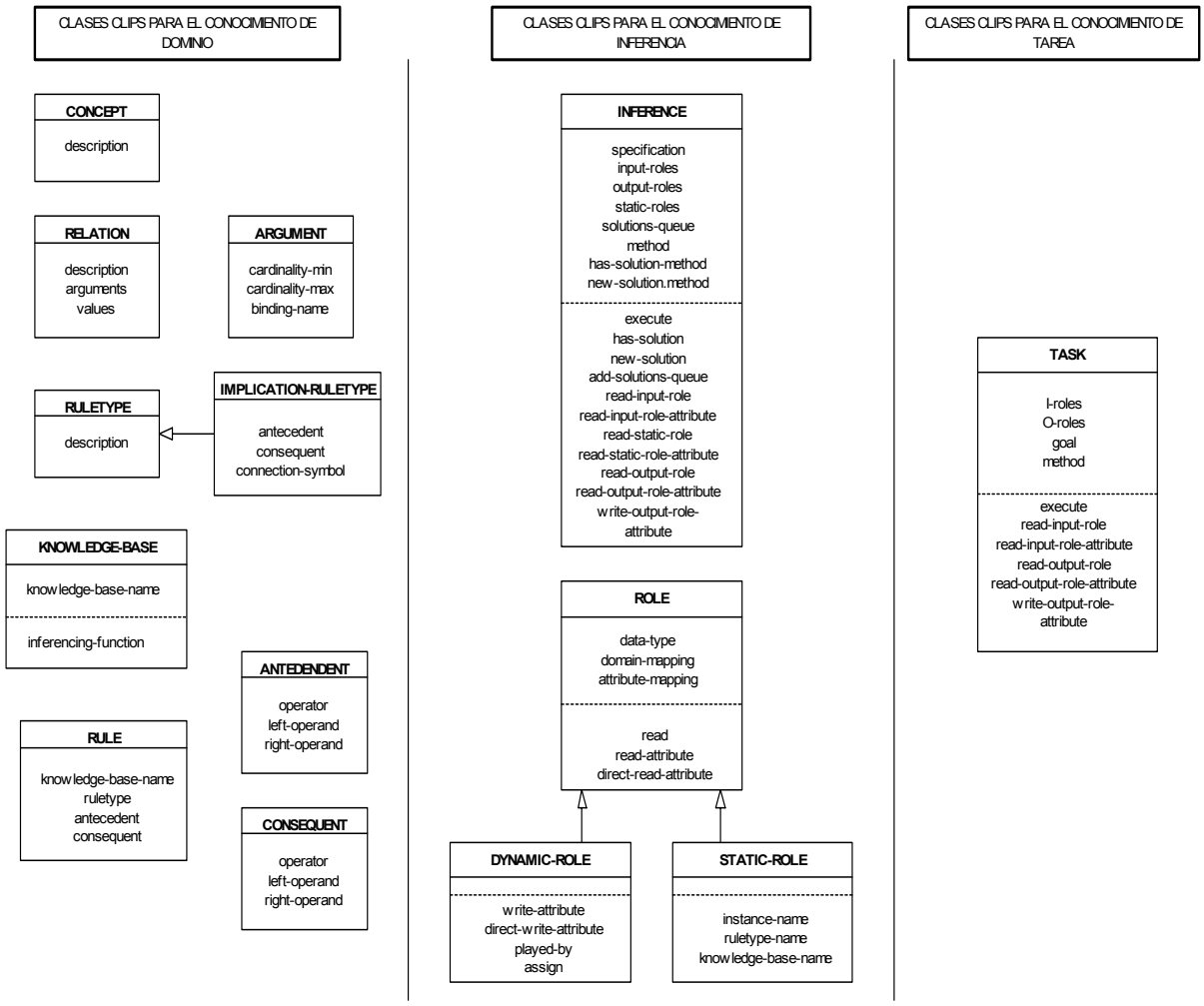

Figura 2.10. Clases CLIPS empleadas para representar los objetos de CommonKADS

Implementación de Objetos del Conocimiento de Dominio

- Concepto

En la sección anterior ya se definió la noción de Concepto. Veamos su equivalente en código CLIPS.

Definición CML:

```
CONCEPT <nombre-concepto> ;
   DESCRIPTION : "descripción del concepto";
   [SUB-TYPE-OF : <nombre-concepto> ;]
```

```
   ATTRIBUTES :
      <nombre-atributo> : <tipo> ;
      [CARDINALITY : <cardinalidad> ;]
      <nombre-atributo> : <tipo> ;
      [CARDINALITY : <cardinalidad> ;]
      ...
END CONCEPT <nombre-concepto> ;
```

Código CLIPS:

```
   (defclass CONCEPT (is-a USER) (role abstract)
      (slot description)
   )
```

Ejemplos:

```
(defclass Estado-Coche (is-a CONCEPT) (role concrete) (pattern-match reactive)
    (slot description (default "estado del coche"))
    (slot estado-general (create-accessor read-write))
)

(defclass Batería (is-a Estado-Coche) (role concrete) (pattern-match reactive)
    (slot description (default "estado de la batería del coche"))
    (slot estado (create-accessor read-write))
)

(defclass Aceite (is-a Estado-Coche) (role concrete) (pattern-match reactive)
    (slot description (default "estado del aceite del coche"))
    (slot nivel (create-accessor read-write))
)
(defclass Motor (is-a Estado-Coche) (role concrete) (pattern-match reactive)
    (slot description (default "estado del motor del coche"))
    (slot estado (create-accessor read-write))
)
```

Se ha definido, por lo tanto, una clase CLIPS llamada CONCEPT, que servirá de base para la implementación de cualquier Concepto CommonKADS. Así, el Concepto Estado-Coche aparece como una clase hija de ésta (is-a), y además introduce el Atributo estado-general como un slot de la clase. La propiedad create-accessor read-

write (facet) de los slots, permite que estos puedan ser leídos y modificados. Por otro lado, la propiedad pattern-match-reactive, de la clase, la capacita para que sus instancias puedan ser utilizadas como antecedentes de reglas del CLIPS.

Tenemos entonces que la clase CONCEPT sirve más bien como referencia para la definición de los Conceptos. De hecho, al ser una clase abstracta (role abstract), no es posible crear instancias directamente de ella.

En lo que se refiere a la construcción Subtipo de, se observa como se ha aprovechado el mecanismo de herencia entre clases, propio de la programación orientada a objetos, para su implementación. De esta forma, los Conceptos Batería y Aceite heredan el slot estado-general de la clase padre Estado-Coche.

- Relación

Definición CML:

```
RELATION <nombre-relacion> ;
   DESCRIPTION : "descripción de la relación" ;
   ARGUMENTS : <nombre-concepto>, <nombre-concepto>, ...;
      [CARDINALITY : <cardinalidad> ;]
   ATTRIBUTES :
      <nombre-atributo> : <tipo> ;
      [CARDINALITY : <cardinalidad> ;]
      <nombre-atributo> : <tipo> ;
      [CARDINALITY : <cardinalidad> ;]
      ...
END RELATION <nombre-relacion> ;
```

Código CLIPS:

```
(defclass RELATION (is-a USER) (role abstract)
   (slot description (create-accessor read-write))
   (multislot arguments (create-accessor read-write))
   (multislot values (create-accessor read-write))
)

(defclass ARGUMENT (is-a USER) (role concrete)
   (slot binding-name (create-accessor read-write))
   (slot cardinality-min (create-accessor read-write))
```

```
    (slot cardinality-max (create-accessor read-write))
)
```

Ejemplos:

```
(defclass Coche (is-a CONCEPT) (role concrete) (pattern-match reactive)
    (slot description (default "vehículo terrestre"))
)

(defclass Persona (is-a CONCEPT) (role concrete) (pattern-match reactive)
    (slot description (default "persona"))
)

(make-instance Coche-pertenece-Persona of ARGUMENT
    (binding-name Coche)
    (cardinality-min 0)
    (cardinality-max -1)
)

(make-instance Persona-posee-Coche of ARGUMENT
    (binding-name Persona)
    (cardinality-min 0)
    (cardinality-max 1)
)

(defclass pertenecer-a (is-a RELATION) (role concrete) (pattern-match reactive)
    (slot description (default "relación de pertenencia entre personas y coches"))
    (multislot arguments (default Coche-pertenece-Persona Persona-posee-Coche) (create-
accessor read-write))
)
```

Para implementar en CLIPS una Relación entre dos conceptos, se han definido dos nuevas clases, la clase RELATION y la clase ARGUMENT.

La clase ARGUMENT se define como apoyo a la implentación de RELATION (y de otros objetos como veremos posteriormente). Se utiliza para identificar los Conceptos que intervienen en la Relación. Así, en la Relación del ejemplo, participan los Conceptos Coche y Persona, y puede observarse, como se han creado dos instancias de ARGUMENT asociadas a cada uno de ellos. El slot binding-name contiene el

nombre del Concepto, y los slots cardinality-min y cardinality-max, indican su participación en la Relación (-1 equivale a 0+).

Por su parte, la clase RELATION incluye un multislot arguments donde se guardarán las instancias creadas de la clase ARGUMENT.

- Tipo de Regla

Definición CML:

```
RULE-TYPE <nombre-ruletype> ;
   DESCRIPTION : "descripción del ruletype" ;
   ANTECEDENT : <nombre-concepto>, <nombre-concepto>, ...;
      [CARDINALITY : <cardinalidad> ;]
   CONSEQUENT : <nombre-concepto> ;
      [CARDINALITY : <cardinalidad> ;]
   CONNECTION-SYMBOL : <símbolo-conector> ;
END RULE-TYPE <nombre-ruletype> ;
```

Código CLIPS:

```
(defclass RULETYPE (is-a USER) (role abstract)
   (slot description      (create-accessor read-write))
)

(defclass IMPLICATION-RULETYPE (is-a RULETYPE) (role concrete)
   (multislot antecedent (create-accessor read-write))
   (slot consequent (create-accessor read-write))
   (slot connection-symbol (create-accessor read-write))
)
```

Ejemplos:

```
(make-instance estado-1 of ARGUMENT
   (binding-name Estado-Coche)
   (cardinality-min 1)
   (cardinality-max -1)
)
```

```
(make-instance estado-2 of ARGUMENT
    (binding-name Estado-Coche)
    (cardinality-min 1)
    (cardinality-max -1)
)

(make-instance Dependencia-Estados of IMPLICATION-RULETYPE
    (description "dependencia entre estados en los que puede encontrarse un paciente")
    (antecedent estado-1)
    (consequent estado-2)
    (connection-symbol causa)
)
```

De nuevo vuelve a usarse la clase ARGUMENT para especificar el Antecedente y el Consecuente del Tipo de Regla definido. La clase IMPLICATION-RULETYPE se ha definido a partir de la clase abstracta más general RULETYPE. Esto se debe a que existe otro Tipo de Regla de restricción (constraint), de la cual no nos vamos a ocupar por su limitada aplicabilidad. Hay que destacar también que al definir antecedent como multislot, se da la posibilidad de referenciar Atributos de diferentes Conceptos en el Antecedente de un Tipo de Regla.

- Base de Conocimiento

Definición CML:

```
KNOWLEDGE-BASE <nombre-knowledgebase> ;
   USES : <nombre-ruletype> FROM <nombre-domainSchema>;
      EXPRESSIONS :
         <expresiones>  ==<símbolo-conectivo>=>  <expresiones> ;
         <expresiones>  ==<símbolo-conectivo>=>  <expresiones> ;
         ...
END RULE-TYPE <nombre-knowledgebase> ;
```

Código CLIPS:

```
(defclass KNOWLEDGE-BASE (is-a USER) (role concrete)
   (slot knowledge-base-name (create-accessor read-write))
```

```
)

(defmessage-handler KNOWLEDGE-BASE inferencing-function primary (?inference-module-
name))

(defclass RULE (is-a USER) (role concrete) (pattern-match reactive)
    (slot knowledge-base-name (create-accessor read-write))
    (slot ruletype (create-accessor read-write))
    (slot antecedent (create-accessor read-write))
    (slot consequent (create-accessor read-write))
)

(defclass ANTECEDENT (is-a USER) (role concrete) (pattern-match reactive)
    (slot operator (create-accessor read-write))
    (slot left-operand (create-accessor read-write))
    (slot right-operand (create-accessor read-write))
)

(defclass CONSEQUENT (is-a USER) (role concrete) (pattern-match reactive)
    (slot operator (create-accessor read-write))
    (slot left-operand (create-accessor read-write))
    (slot right-operand (create-accessor read-write))
)
```

Ejemplos:

```
(make-instance Relaciones-Causa-Efecto-Coche of KNOWLEDGE-BASE
    (knowledge-base-name Relaciones-Causa-Efecto-Coche)
)

(make-instance regla-1 of RULE
    (knowledge-base-name Relaciones-Causa-Efecto-Coche)
    (ruletype Dependencia-Estados)
    (antecedent antecedente-regla)
    (consequent consecuente-regla)
)

(make-instance antecedente-regla of ANTECEDENT
(operator or)
(left-operand antedecente-regla-i)
(right-operand antecedente-regla-d)
)

(make-instance antecedente-regla-i of ANTECEDENT
(operator =)
```

```
(left-operand Batería.estado)
(right-operand bajo)
)

(make-instance antecedente-regla-d of ANTECEDENT
(operator =)
(left-operand Aceite.nivel)
(right-operand bajo)
)

(make-instance consecuente-regla of CONSEQUENT
(operator =)
(left-operand Motor.estado)
(right-operand no-arranca)
)
```

En primer lugar tenemos la clase KNOWLEDGE-BASE, cuyas instancias permitirán especificar las Bases de Conocimiento definidas. Para implementar las Expresiones se utiliza la clase RULE, que tiene un slot para indicar el nombre de la Base de Conocimiento con la que se trabaja, otro slot que indica el Tipo de Regla y dos slots adicionales para el antecedente y el consecuente, respectivamente.

Como se desprende del ejemplo, el mecanismo implementado para pasar las Expresiones a CLIPS es muy general y potente, ya que permite tratar con reglas muy complejas que involucran gran cantidad de subexpresiones conectadas entre si a través de operadores lógicos. Básicamente, consiste en ir formando un árbol binario con todos los elementos que componen la Expresión.

Además se ha definido el método inferencing-function asociado a la clase KNOWLEDGE-BASE, que será usado por las Inferencias para acceder a las Expresiones o reglas de una determinada Base de Conocimiento. Su funcionamiento es como sigue:

1º Pone como módulo de trabajo, el módulo de la Inferencia.

2º Elimina información innecesaria, producto de anteriores ejecuciones de la Inferencia.

3º Activa las reglas necesarias y las ejecuta.

4º Pone como módulo de trabajo, el que había antes de la ejecución de la Inferencia.

Hay que aclarar que cuando se habla de módulo, nos referimos a los módulos de CLIPS, que permiten aislar el ámbito de las reglas con las que trabaja cada inferencia.

Finalmente, hay que añadir que además de Expresiones, también es posible trabajar con Bases de Conocimiento que contengan Instancias (INSTANCES).

Implementación de Objetos del Conocimiento de Inferencia

- Roles de Conocimiento

Definición CML:

```
KNOWLEDGE-ROLE <nombre-rol> ;
   DESCRIPTION : "descripción del rol" ;
   TYPE : static | dynamic ;
   DOMAIN-MAPPING : <nombre-del-objeto-que-juega-el-rol> ;
END KNOWLEDGE-ROLE <nombre-rol> ;
```

Código CLIPS:

```
(defclass ROLE (is-a USER) (role abstract)
   (slot data-type (create-accessor read-write))
   (multislot domain-mapping (create-accessor read-write))
   (multislot attribute-mapping (create-accessor read-write))
)

(defmessage-handler ROLE read primary ())
(defmessage-handler ROLE direct-read-attribute primary (?atributo))
(defmessage-handler ROLE read-attribute primary (?atributo))

(defclass STATIC-ROLE (is-a ROLE) (role concrete))

(defmessage-handler STATIC-ROLE instance-name primary ())
(defmessage-handler STATIC-ROLE ruletype-name primary ())
(defmessage-handler STATIC-ROLE knowledge-base-name primary ())

(defclass DYNAMIC-ROLE (is-a ROLE) (role concrete))

(defmessage-handler DYNAMIC-ROLE direct-write-attribute primary (?atributo $?valor))
```

```
(defmessage-handler DYNAMIC-ROLE write-attribute primary (?atributo $?valor))
(defmessage-handler DYNAMIC-ROLE played-by primary ($?name-instance))
(defmessage-handler DYNAMIC-ROLE assign (?source-role))
```

Ejemplos:

```
(make-instance Modelo-Causal of STATIC-ROLE
    (data-type IMPLICATION-RULETYPE)
    (domain-mapping Dependencia-Estados FROM Relaciones-Causa-Efecto-Coche)
)

(make-instance Fallo-Observado of DYNAMIC-ROLE
    (data-type Estado-Coche)
)

(make-instance Fallo-1 of Motor
    (estado no-arranca)
)

(send [Fallo-Observado] played-by Fallo-1
```

Los Roles Dinámicos y los Roles Estáticos se representan a través de las clases DYNAMIC-ROLE y STATIC-ROLE, respectivamente, que provienen de la clase general abstracta ROLE. El slot data-type se refiere a la clase, cuya instancia va a jugar el papel definido por el Rol. El multislot domain-mapping apunta a esa instancia concreta. Finalmente, el multislot attribute-mapping identifica los atributos de la instancia actual que serán utilizados como los atributos de la clase en data-type. Con esto, se consigue que una instancia pueda desempeñar un determinado Rol, aunque su clase no coincida con la especificada en el slot data-type. De esta manera, Roles que operan con instancias de una clase determinada, pueden trabajar también con instancias de otras clases, permitiendo así la reusabilidad del código.

Se puede observar que se han definido numerosos métodos, algunos comunes a los dos tipos de Roles, y otros particulares de cada uno de ellos. Estos métodos están orientados, principalmente, a relacionar los Roles con los objetos del dominio. Concretamente, el método played-by definido para los Roles Dinámicos es especialmente flexible y potente. En esencia, se encarga de unir el Rol con la instancia

que, en un momento dado, pasa a jugar el papel indicado. Se han contemplado las siguientes situaciones:

1ª- La instancia ya existe y es de la clase especificada en el slot data-type. Sería el caso que se muestra en los ejemplos y representa la situación más sencilla.

2ª- Se quiere crear la instancia al mismo tiempo que es asignada al Rol correspondiente. En este caso, la sintaxis sería:

```
(send [nombre-rol] played-by new-instance nombre-concepto
[ atributo-1 valor-1 ]
[ atributo-2 valor-2 ]
...
)
```

3ª- La instancia ya existe, pero la clase a la que pertenece no corresponde con la especificada en el slot data-type. En este caso, será necesario indicar, cómo se emparejan los atributos de las dos clases:

```
(send [nombre-rol] played-by nombre-instancia
[ atributo-1 atributo-2 ... ]
into
[ atributo-del-data-type-1 atributo-del-data-type-2 ... ]
)
```

4ª- Se dan conjuntamente las situaciones 2ª y 3ª:

```
(send [nombre-rol] played-by new-instance nombre-instancia
[ atributo-1 into atributo-del-data-type-1 valor-1]
[ atributo-2 valor-2]
[ atributo-3 into atributo-del-data-type-2 valor-3]
...
)
```

- Inferencia

Definición CML:

INFERENCE *<nombre-inferencia>*;
SPECIFICATION : *"descripción de la inferencia"*;
ROLES :

```
        INPUT: <nombre-rol>, <nombre-rol>, …;
        OUTPUT: <nombre-rol>, <nombre-rol>, …;
        STATIC: <nombre-rol>, <nombre-rol>, …;
END INFERENCE <nombre-inferencia>;
```

Código CLIPS:

```
(defclass INFERENCE (is-a USER) (role concrete) (pattern-match reactive)
    (slot specification (create-accessor read-write))
    (multislot input-roles (create-accessor read-write))
    (multislot output-roles (create-accessor read-write))
    (multislot static-roles (create-accessor read-write))
    (multislot solutions-queue (create-accessor read-write))
    (slot method (create-accessor read-write))
    (slot has-solution-method (create-accessor read-write))
    (slot new-solution-method (create-accessor read-write))
)

(defmessage-handler INFERENCE execute primary ($?roles))
(defmessage-handler INFERENCE has-solution primary ())
(defmessage-handler INFERENCE new-solution primary ())
(defmessage-handler INFERENCE add-solutions-queue primary (?new-solution ?position))
(defmessage-handler INFERENCE read-input-role (?posicion))
(defmessage-handler INFERENCE read-input-role-attribute (?posicion ?atributo))
(defmessage-handler INFERENCE read-static-role (?posicion))
(defmessage-handler INFERENCE read-static-role-attribute (?posicion ?atributo))
(defmessage-handler INFERENCE read-output-role (?posicion))
(defmessage-handler INFERENCE read-output-role-attribute (?posicion ?atributo))
(defmessage-handler INFERENCE write-output-role-attribute (?posicion ?atributo $?values))
```

Ejemplos:

```
(make-instance Buscar-Causa of INFERENCE
    (specification "buscar la causa que pudo haber provocado los síntomas")
    (input-roles Fallo-Observado)
    (output-roles Hipótesis)
    (static-roles Modelo-Causal)
    (method Buscar-Causa)
    (has-solution-method general-has-solution)
    (new-solution-method general-new-solution)
)

(send [Buscar-Causa] execute)
```

La clase INFERENCE implementa las Inferencias del Modelo de Conocimiento. El slot specification es meramente descriptivo. Los multislots input-roles, output-roles y static-roles, contienen los respectivos Roles asociados a la Inferencia. De la forma en que se implementan estos Roles, nos ocuparemos a continuación. En el multislot solutions-queue se guardan las soluciones como resultado de la ejecución de la Inferencia. Existen también tres slots que almacenan las funciones de CLIPS asociadas a la Inferencia. El slot method contiene el nombre de la función que ejecutará la Inferencia en sí. Por otro lado, los slots has-solution-method y new-solution-method, se refieren a funciones que manejan la cola de soluciones. En el ejemplo, apuntan a dos funciones generales general-has-solution y general-new-solution.

También para las Inferencias, se han definido un gran número de métodos que facilitan su manejo. De todos ellos, sin duda, el más importante es el método execute, que desencadena la ejecución de la inferencia. Dada su importancia, pasamos a comentarlo en más detalle.

La llamada a este método puede realizarse de dos maneras:

1ª- Si los Roles de entrada y salida apuntan a las instancias correctas, antes y después de la llamada, la sintaxis sería simplemente:

```
(send [nombre-inferencia] execute)
```

2ª- Si se requiere que los Roles de entrada apunten a instancias asignadas a otros Roles, o que la instancia apuntada por los Roles de salida sea asignada a otro Rol, la sintaxis quedaría como:

```
(send [nombre-inferencia] execute
    [ input-role-1 <-- [role-A] ]
    [ input-role-3 <-- [role-B] ]
    [ output-role-7 --> [role-C] ]
)
```

En cualquiera de los casos, lo más importante a destacar es el hecho de que el código CLIPS para la ejecución de una Inferencia, se reduce a una llamada a un método asociado a la clase INFERENCE.

Para terminar con las Inferencias, indicamos brevemente lo que ocurre cuando se produce una llamada al método execute.

1º La Inferencia correspondiente llama a la función cuyo nombre se encuentra en el slot method.

2º La función llamada se encarga de ejecutar la Inferencia, obteniendo los datos necesarios de los Roles.

3º Todas las soluciones obtenidas se almacenan en el slot solutions-queue, y una de ellas en el Rol de salida.

- Tarea y Método de Tarea

Definición CML:

```
TASK <nombre-tarea>;
    GOAL : "descripción del objetivo de la tarea";
    ROLES :
        INPUT: <nombre-rol>, <nombre-rol>, …;
        OUTPUT: <nombre-rol>, <nombre-rol>, …;
END TASK <nombre- tarea>;
```

Código CLIPS:

```
(defclass TASK (is-a USER) (role concrete)
   (multislot I-roles (create-accessor read-write))
   (multislot O-roles (create-accessor read-write))
   (slot goal (create-accessor read-write))
   (slot method (create-accessor read-write))
)

(defmessage-handler TASK execute primary ())
(defmessage-handler TASK read-input-role (?posicion))
(defmessage-handler TASK read-input-role-attribute (?posicion ?atributo))
(defmessage-handler TASK read-output-role (?posicion)
(defmessage-handler TASK read-output-role-attribute (?posicion ?atributo))
(defmessage-handler TASK write-output-role-attribute (?posicion ?atributo $?values))
```

Ejemplos:

```
(make-instance Diagnóstico-Averías-Coche of TASK
    (I-roles Fallo-Observado)
    (O-roles Causa-Probable Evidencia)
    (goal "Encontrar una causa probable para la avería")
    (method Diagnóstico-usando-generar-y-comprobar)
```

La clase TASK contiene slots para indicar el objetivo, los Roles de entrada y salida, y la función de CLIPS que se encargará de la ejecución de la tarea. Esta función consistirá, básicamente, en llamadas a ejecuciones de Inferencias (como en el punto anterior), controladas por sentencias de control de CLIPS.

Se han definido también una serie de métodos, similares a los de las Inferencias, de tal forma que el código que se genere, necesariamente, tendrá que empezar de la siguiente manera:

(send [nombre-tarea] execute)

2.2.1.3 Librería de Apoyo: Manipulación de Listas

El lenguaje CLIPS permite trabajar con listas, pero estas listas a su vez no pueden contener otras listas. Por este motivo, se creó una librería llamada ***list.bat***, con funciones que ofrecen esta posibilidad.

2.2.1.4 Librerías de Apoyo Generales

Se trata de librerías que contienen objetos y funciones CLIPS, que se consideran tan generales que deben ser siempre incluidos en cualquier aplicación que se haga. Las librerías de las que hablamos son ***DS-general-domain-schema.bat, KB-general.bat, y UF-general.bat.***

Para dar una idea de su contenido, mostramos la ***DS-general-domain-schema.bat***:

```
(defclass TRUTH-VALUE (is-a CONCEPT) (role concrete) (pattern-match reactive)
    (slot description (default "true or false"))
    (slot value (create-accessor read-write))
)

(defclass LIST (is-a CONCEPT) (role concrete) (pattern-match reactive)
```

```
    (slot description (default "ordered collection of items"))
    (multislot items (create-accessor read-write))
)

(defclass NODE (is-a CONCEPT) (role concrete) (pattern-match reactive)
    (slot description (default "item for the LISTs"))
    (slot key (create-accessor read-write))
)
```

Como puede verse, se trata de tres clases CONCEPT. La primera sirve para representar el valor verdadero-falso, y las otras dos para definir listas y nodos de listas, que serán manipuladas con ayuda de la librería descrita en el apartado anterior.

2.2.1.5 Analizador de Reglas

Debido a la complejidad que pueden llegar a tener las Expresiones que componen una Base de Conocimiento, se ha desarrollado una herramienta de ayuda para facilitar su análisis. Es importante tener claro que estas Expresiones no se convierten a lo que en CLIPS se entiende por reglas (que serían gestionadas directamente por el motor de inferencia de éste), sino a instancias de la clase RULE, como ya se explicó en una apartado anterior.

En esencia, lo que hace el analizador de reglas, es ir analizando estas instancias, e ir generando hechos (facts) de CLIPS que sirven como antecedentes y consecuentes de "verdaderas" reglas CLIPS.

2.2.2 Apoyo a la Traducción CommonKADS-CLIPS

En la sección anterior se expuso la arquitectura software propuesta para la implementación del Modelo de Conocimiento de CommonKADS. Básicamente se trataba de una serie de librerías que servían de soporte a la implementación, tratando de facilitar la labor del usuario. A pesar de todo, es evidente la complejidad del proceso de traducción "a mano" de los objetos del CommonKADS a CLIPS.

Por este motivo, se planteó la creación de una herramienta que automatizara parte de este proceso, liberando así al usuario de parte del trabajo de traducción. Obviamente, no todo resulta igual de fácil de automatizar. La herramienta desarrollada

es capaz de traducir el código CML de entrada, excepto los cuerpos de las Inferencias y Tareas (lo que correspondería al código de las funciones CLIPS apuntadas en el slot method de las clases INFERENCE y TASK). En cualquier caso, no cabe duda de que supone una gran ayuda.

El traductor se desarrolló en entorno Linux, utilizando las herramientas: ***gcc***, ***flex*** [Paxson 95] y ***bison*** [Donelly 95].

gcc es el compilador de C estándar en Linux, y con él se creará el ejecutable final disponible para el usuario.

Flex y ***bison*** son herramientas que sirven para la construcción de compiladores e intérpretes. ***Flex*** es un analizador léxico, capaz de generar un programa que realiza *pattern-matching* de los elementos léxicos que componen el CML. Por su parte, ***bison*** genera un analizador sintáctico que realiza la acción programada cuando encuentra una construcción sintáctica.

Ambos permiten la generación del programa traductor que finalmente convierte el código CML a CLIPS. En el apéndice se incluyen algunos de ficheros generados automáticamente por el traductor.

En la figura 2.11, se esquematiza el proceso de creación del traductor.

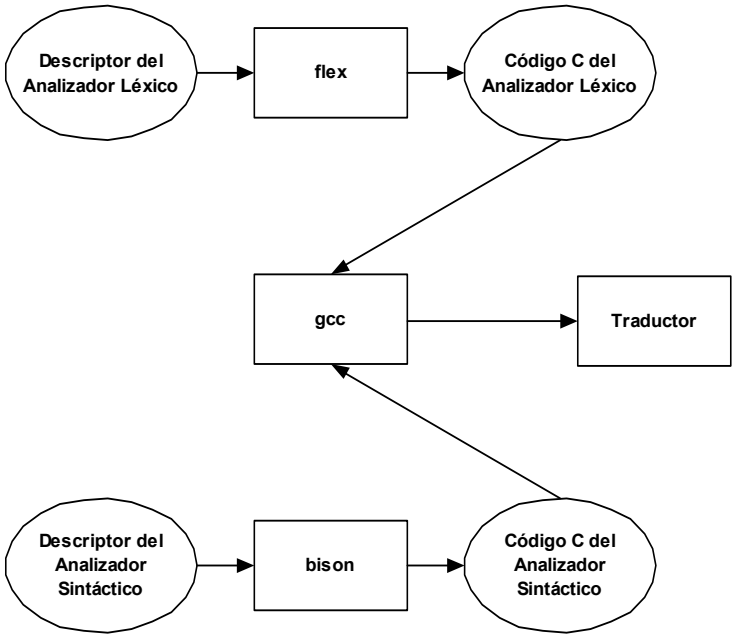

Figura 2.11. Esquema de creación del traductor CommonKADS-CLIPS

CAPÍTULO 3. VERIFICACIÓN DE LA NORMALIDAD DE UN CONJUNTO DE DATOS

En campos tales como la ciencia, la ingeniería o la economía, está muy extendido el uso de modelos estadísticos y de distribuciones de probabilidad. El problema de la bondad del ajuste consiste en evaluar lo bien que se ajustan los datos observados a un modelo probabilístico dado.

Una manera informal de llevar esto a cabo sería dibujar un histograma mostrando las frecuencias de distribución de los datos y juzgar "a ojo" su parecido con la función de distribución teórica. Tales métodos gráficos son bastante subjetivos y no parecen los más recomendables para hacer una valoración fiable, si bien pueden ser un buen complemento a otro tipo de métodos. Para tratar de conseguir esta objetividad, se suele plantear el problema como un contraste de hipótesis clásico.

De especial interés es el problema de determinar si los datos se ajustan a una distribución normal y es de hecho en lo que se centra este capítulo.

Es incuestionable el importante papel que la distribución normal juega como modelo de referencia en multitud de aplicaciones. En este trabajo estamos interesados, en particular, en las aplicaciones relacionadas con la clasificación de patrones, donde la distribución normal suele utilizarse como modelo paramétrico simple de las distribuciones de probabilidad de las clases. Esto supone una simplificación importante en el problema del diseño de clasificadores [Duda 73].

Como alternativas a los procedimientos clásicos de determinación de la normalidad, se proponen dos aproximaciones al problema desde un contexto bayesiano. Una, basada en la utilización de redes neuronales como medio para combinar estadísticos habitualmente empleados en la construcción de tests de hipótesis convencionales. La otra, basada en la Teoría de Grandes Desviaciones, sirve como base para la selección de un número reducido de distribuciones diferentes de la normal con las cuales se diseña el procedimiento de test.

3.1 Teoría de Verificación de las Hipótesis

En esta sección se explican brevemente los fundamentos de la teoría de verificación de hipótesis tanto en el contexto clásico, como en el bayesiano. Empezaremos considerando las hipótesis simples para después pasar a las compuestas.

3.1.1 Verificación de un Número Finito de Hipótesis Simples

Llamaremos hipótesis simple a cualquier suposición acerca de la distribución de una muestra X, que defina de forma unívoca esta distribución.

Si tenemos r distribuciones $P_1,\ldots,P_r$, y sabemos que X proviene de una de ellas, el problema consiste en determinar de cual se trata. Esto da lugar a r hipótesis simples de la forma $H_1,\ldots,H_r$.

Se denomina criterio estadístico para verificar r hipótesis simples $H_1,\ldots,H_r$, a toda función medible $\psi : \chi^N -> \{H_1,\ldots,H_r\}$, donde χ^N es el espacio muestral de muestras de tamaño N. A esta función también se le denomina regla de decisión, test o contraste de hipótesis.

La calidad de un criterio viene dada, a menudo, por el conjunto de probabilidades de decisiones erróneas:

$$P_{E_i} = P_i\left(\psi(X) \neq H_i\right) \tag{3.1}$$

donde α_i representa la probabilidad de rechazar la hipótesis H_i, cuando ésta es cierta. Es lo que se denomina probabilidad del error tipo i del criterio ψ.Lo ideal, será entonces escoger ψ, de forma que estos errores sean lo más pequeños posibles.

Para poder comparar diferentes criterios, debemos de buscar alguna forma de ordenarlos. Una posible opción es considerar que el criterio ψ_1 es mejor que el ψ_2, si se cumple:

$$P_{E_i}\left(\psi_1\right) \leq P_{E_i}\left(\psi_2\right), \quad i = 1,\ldots,r \tag{3.2}$$

y al menos para algún valor de i, se da la desigualdad estricta.

Sin embargo, este criterio no siempre es aplicable, y se hace necesario restringir el conjunto de reglas de decisión a considerar. En particular, vamos a considerar la clase de criterios que cumple:

$$K_{\alpha_1,\ldots,\alpha_{r-1}} = \left\{\psi : P_{E_j}\left(\psi\right) = \alpha_j;\ j = 1,2,\ldots,r-1\right\} \tag{3.3}$$

esto es, fijamos todos las probabilidades de error a un valor determinado, excepto una de ellas.

El criterio $\psi_0 \in K_{\alpha_1,\ldots,\alpha_{r-1}}$ se llama criterio más potente en esta clase, si para cualquier otro criterio ψ perteneciente a la clase, se cumple:

$$P_{E_r}\left(\psi_0\right) \leq P_{E_e}\left(\psi\right) \tag{3.4}$$

A la cantidad $1 - P_{E_r}\left(\psi\right)$, es a lo que se llama la potencia del test. Esta es la forma habitual de ordenar los criterios en el enfoque clásico del problema.

En el enfoque bayesiano, también se establece una relación de orden entre las diferentes reglas de decisión, pero para introducirla es preciso definir antes algunos conceptos.

En primer lugar, se asume que la distribución P_i, de la que fue extraída la muestra X, se ha elegido aleatoriamente. En este caso, las hipótesis $H_i = \left\{X \in P_i\right\}, \quad i = 1,\ldots r$ serán sucesos aleatorios, y designaremos las probabilidades de estos sucesos por $P\left(H_i\right)$. A estas probabilidades se les denomina probabilidades a

priori. Con esto resulta sencillo comparar los criterios a partir de la probabilidad media $P_E(\psi)$ de error del criterio ψ:

$$P_E(\psi) = \sum_{i=1}^{r} P(H_i) P_i(\psi(X) \neq H_i) \tag{3.5}$$

ordenándolos según la magnitud de $P_E(\psi)$.

El criterio ψ se denominará bayesiano para unas probabilidades a priori dadas, si para estas probabilidades se minimiza la probabilidad promedio de error.

Para una muestra X, en el contexto bayesiano, se pueden calcular las probabilidades a posteriori, sin más que aplicar la regla de Bayes:

$$P(H_i \mid X) = \frac{P(H_i) p(X \mid H_i)}{p(X)} \tag{3.6}$$

donde $p(X \mid H_i)$ es la densidad de probabilidad de X, supuesta cierta la hipótesis H_i, y $p(X)$ es la densidad incondicional.

Puede demostrarse fácilmente que si se toma como regla de decisión, aceptar la hipótesis cuya probabilidad a posteriori sea la mayor, el criterio resultante es bayesiano [Borovkov 88].

3.1.2 Verificación de Hipótesis Compuestas

Una vez explicadas las cuestiones fundamentales relativas a la verificación de hipótesis simples, hacemos una breve introducción a las compuestas, cuyo tratamiento resulta mucho más complicado.

En primer lugar hay que decir que consideraremos como hipótesis compuesta, a toda aquella que no sea simple. Normalmente, suelen considerarse sólo dos hipótesis a las que se denomina hipótesis nula H_o, e hipótesis alternativa H_1. La hipótesis nula suele jugar el papel de hipótesis fundamental, mientras que la alternativa se ve más bien como desviaciones respecto a ésta.

Realmente, no vamos a profundizar demasiado en este tema porque sería excesivamente largo y no demasiado interesante para nuestros propósitos. Hay que tener en cuenta que en las técnicas que se han desarrollado, se han hecho una serie de simplificaciones que convierten al problema, casi en una verificación de hipótesis simples. Conviene, no obstante, aclarar que la verificación de hipótesis compuestas en

el contexto clásico, no tiene, en general una solución óptima. El ideal es tratar de diseñar un test uniformemente más potente, el cual implica que se cumpla la condición (3.4) para cada una de las hipótesis simples que forman H_1. Se trata de una condición muy fuerte y difícil de alcanzar en la práctica, por lo que, frecuentemente, hay que relajarla para poder llegar a alguna solución, aunque sea subóptima [Borovkov 88].

En el contexto bayesiano, se hace preciso introducir una función de *prior* que suele plantear muchas complicaciones, excepto en el supuesto en el que las hipótesis se refieran a un conjunto discreto de distribuciones. Este es precisamente el caso que se nos presentará más adelante en este capítulo, y por eso conviene que nos detengamos un poco en él.

Supongamos, entonces, que tenemos un problema de verificación de hipótesis donde la hipótesis H_o supone que X proviene de una cualquiera del conjunto de distribuciones $P_{0j}\ j=1,\ldots m$, esto es, $H_0=\{X\in P_{0j}\},\quad j=1,\ldots m$, y análogamente, $H_1=\{X\in P_{1j}\},\quad j=1,\ldots n$. Designando por $P(H_i)\quad i=0,1$, a las probabilidades a priori de las hipótesis y por $P(H_{ij})\quad i=0,1$, la función de *prior* correspondiente, podemos escribir ahora la probabilidad a posteriori de la hipótesis H_o, de la siguiente manera:

$$P(H_0\mid X)=\frac{P(H_0)\sum_{j=1}^{m}P(H_{0j})p(X\mid H_0)}{p(X)} \tag{3.7}$$

y análogamente para H_1. De resto, todo lo aplicable a hipótesis simples sigue siendo válido.

En el caso más general en el que el espacio de distribuciones varía de forma continua, resulta difícil especificar la función de *prior* más adecuada para un problema dado, y se han propuesto múltiples soluciones [Berger 85], [Bernardo 00]

3.2 Tratamiento Clásico del Problema de la Bondad del Ajuste

Formalmente, podemos definir un test de bondad del ajuste de la siguiente forma:

Dada una muestra aleatoria $X_1, X_2, \ldots, X_N$, formulamos la hipótesis nula de que la población correspondiente a la muestra tiene función de distribución acumulativa $F(x;\theta), \quad \theta \in \Theta$, frente a la hipótesis alternativa de que la función de distribución acumulativa es $G(x;\omega), \quad w \in \Omega$.

Aunque la definición anterior es aplicable a variables vectoriales, es importante aclarar, que en lo que sigue, se tratará siempre el problema monodimensional. Los espacios de parámetros Θ y Ω si serán, en general, multidimensionales y las hipótesis podrán ser simples o compuestas, dependiendo de si especifican completa o parcialmente las distribuciones implicadas. Es importante destacar que, en la práctica, la hipótesis alternativa vendrá especificada, muy a menudo, como cualquier distribución que no esté contenida en la hipótesis nula. Así, si por ejemplo, se plantea un test para determinar si ciertos datos siguen una distribución uniforme, la hipótesis alternativa será cualquier distribución no uniforme. Esta falta de precisión en la definición de las distribuciones alternativas, constituye sin duda una dificultad añadida, cuyas implicaciones se discutirán en una sección posterior.

Se suele distinguir también entre tests ómnibus y direccionales. Los tests ómnibus están pensados para tener potencia moderada contra todo tipo de alternativas, mientras que los tests direccionales rechazan bien alternativas específicas, por lo que resultan más informativos, y no resultan tan potentes frente a otras.

Finalmente, es importante hacer hincapié en el hecho de que, en la práctica, la distribución de los datos difícilmente va a coincidir de forma exacta con el modelo propuesto. Sin embargo, bastará con exigir al mismo, que sea lo suficientemente bueno para nuestros propósitos, esto es, que se encuentre dentro de ciertos márgenes de tolerancia que consideremos aceptables.

3.2.1 Hipótesis de Normalidad

Dentro de los problemas de bondad del ajuste existe uno de especial relevancia y es la verificación de la normalidad de un conjunto de datos observados, esto es, si los datos siguen una distribución normal. Esta relevancia es consecuencia del importante papel que esta distribución juega en todas las ramas de la Probabilidad y la Estadística.

Aunque existen muchas aplicaciones donde es interesante este tipo de análisis de los datos, es importante aclarar, que en este trabajo se ha profundizado en este aspecto, fundamentalmente, con vistas a su consideración como modelo paramétrico simple de sistema clasificador, en el contexto del reconocimiento de patrones.

Son muchas las propiedades a destacar de la distribución normal. Mencionaremos aquellas más significativas dentro de este contexto:

- Tiene propiedades analíticas relativamente simples, permitiendo que muchos resultados útiles sean obtenidos explícitamente. Por ejemplo, cualquier momento de la distribución puede ser expresado en función de μ, el vector con el valor esperado de la distribución en cada dimensión, y Σ, la matriz de covarianza.
- El Teorema del Límite Central establece que, bajo circunstancias bastante generales, la media de N variables aleatorias tiende a distribuirse normalmente cuando N tiende a infinito. La condición principal es que ninguna de las varianzas de las variables domine sobre las otras. Una aplicación común de este teorema se refiere a la suma de un conjunto de variables extraídas de forma independiente de la misma distribución. En la práctica, la convergencia tiende a ser muy rápida, de tal manera que para valores del orden de 10, la aproximación a la normalidad puede ser muy buena. Así, podemos esperar que ciertas medidas obtenidas a partir de los datos, que de forma natural aparezcan como la suma de diferentes componentes, se distribuyan aproximadamente como normales.
- Bajo cualquier transformación lineal, no singular, del sistema de coordenadas, la distancia de Mahalanobis mantiene su forma cuadrática definida positiva. Por lo tanto, bajo este tipo de transformaciones, una distribución normal seguirá siendo normal pero con distintos parámetros de media y covarianza.
- Las densidades condicionales de una distribución normal son normales. Análogamente, las densidades marginales son también normales. Esto último pone de manifiesto la importancia de evaluar la normalidad en cada dimensión, siendo condición necesaria para la normalidad multidimensional.
- Existe una transformación lineal que diagonaliza la matriz de covarianza Σ. Esto lleva a un nuevo sistema de coordenadas, basado en los vectores propios de Σ, en el cual las variables son estadísticamente independientes, de tal manera que la función

de densidad multivariable es el producto de las densidades de cada una de las componentes por separado.

- Para valores dados de la media y de la matriz de covarianza, la función de densidad normal maximiza la entropía.

3.2.2 Contrastes de Significación

Los tests de bondad del ajuste se han diseñado tradicionalmente como contrastes de significación (*Pure Significance Tests*) en los que sólo se especifica la hipótesis nula H_0. Para estos contrastes se define un estadístico $T = t(X)$, donde t es una función de los datos observados. Este estadístico deberá cumplir [Cox 74]:

- Su distribución, cuando H_0 es cierta, se conoce al menos de forma aproximada. En particular, si H_0 es compuesta, la distribución de T debería de ser la misma o aproximadamente la misma, para todas las hipótesis simples que constituyen la hipótesis nula.
- Cuanto mayor sea el valor del estadístico, mayor será la evidencia de "alejamiento" de H_0.

Se define también el valor-P P_{obs} para unos datos observados x, como:

$$P_{obs} = pr(T \geq t_{obs}; H_0) \tag{3.8}$$

donde $t_{obs} = t(X)$, esto es, la probabilidad de obtener un resultado tan raro o más que el obtenido, cuando la hipótesis nula es cierta. Este valor-P se suele utilizar en el contexto clásico como una medida de la consistencia de los datos observados con la hipótesis H_0.

El procedimiento habitual consiste en prefijar ciertos niveles de significación (probabilidad de error correspondiente a rechazar la hipótesis cuando es cierta) que podríamos considerar como estándar (0.01, 0.05 o 0.1), y dependiendo de si P_{obs} excede o no, dichos niveles, se aceptará o se rechazará la hipótesis. Aunque no hay ninguna razón de peso que justifique la elección de estos niveles concretos, resultan

convenientes como valores de referencia para comparar, entre diferentes tests, las potencias con las que se rechazan distribuciones alternativas escogidas arbitrariamente.

Del planteamiento anterior, se dejan entrever algunas limitaciones del procedimiento clásico, algunas de las cuales pasamos a comentar.

- La más obvia es la dificultad a la hora de elegir un estadístico adecuado al problema. Si bien el estadístico se selecciona de acuerdo con el tipo de desviación de la normalidad que interesa detectar, sería deseable disponer de un método más preciso en el que aparecieran de forma explícita las alternativas a considerar.
- Otra limitación más seria es la ausencia de un indicador fiable de hasta que punto unos datos concretos apoyan la hipótesis. En este sentido, los valores-P definidos arriba no parecen ofrecer las garantías necesarias como ha sido puesto de manifiesto en numerosas ocasiones por diversos investigadores del área [Berger 85], [Bernardo 00].

Algunos de los estadísticos más utilizados en este tipo de tests son el chi-cuadrado χ^2 y el de Kolmogorov D_n [Cox 74]. Como están pensados para cualquier hipótesis sobre la forma de las distribuciones, resultan ser poco adecuados cuando se trata el problema particular de la normalidad. En este caso, se suelen utilizar otra clase de estadísticos más apropiados, algunos de los cuales describimos brevemente a continuación.

3.2.3 Estadísticos Usados en Tests de Normalidad Monodimensionales

Momentos de la muestra:

Karl Pearson fue el primero en sugerir que las estimaciones muestrales $\sqrt{b_1}$ y b_2 de $\sqrt{\beta_1} = \frac{\mu_3}{\mu_2^{3/2}}$ y $\beta_2 = \frac{\mu_4}{\mu_2^2}$, respectivamente, podían ser utilizadas para describir distribuciones no normales y como base para tests de normalidad.

$\sqrt{\beta_1}$ es una medida de sesgo o asimetría de las distribuciones. Se anula si la distribución es simétrica y su signo indica el sentido del sesgo, positivo hacia la izquierda y negativo hacia la derecha. De todas formas, esta medida debe ser

interpretada con cierta cautela ya que existen casos en los que también puede anularse para distribuciones asimétricas [Alan 94].

β_2 es una medida de kurtosis y su interpretación es un poco más delicada. La distribución normal tiene $\beta_2 = 3$ y para ciertas distribuciones regulares y simétricas se cumple que si $\beta_2 > 3$, esto implica que el área de las colas es mayor y resultan más puntiagudas (leptokúrticas). Análogamente, si $\beta_2 < 3$, el área de las colas se reduce con respecto a la normal y se vuelven más "aplastadas" (mesokúrticas). Hay que decir que en el caso más general, esta interpretación no siempre es válida [Alan 94].

En una sección posterior se mostrarán ejemplos de distribuciones con diferentes valores de sesgo y kurtosis.

Volviendo a las estimaciones muestrales de estos estadísticos, tenemos que:

$$\sqrt{b_1} = \frac{m_3}{m_2^{3/2}} \quad \text{y} \quad b_2 = \frac{m_4}{m_2^2} \tag{3.9}$$

donde

$$m_k = \frac{\sum_{i=1}^{N} \left(X_i - \overline{X}\right)^k}{N} \quad \text{y} \quad \overline{X} = \frac{\sum_{i=1}^{N} X_i}{N} \tag{3.10}$$

Como estadísticos descriptivos, valores de $\sqrt{b_1}$ y b_2 cercanos a 0 y 3, respectivamente, son indicativos de normalidad. De forma más precisa, sus valores esperados bajo la normalidad son 0 y $3(n-1)/(n+1)$. Además, como ya se ha comentado, los signos y magnitudes de estos proporcionan información acerca del tipo de no normalidad, por ejemplo, $\sqrt{b_1} > 0$ corresponde a sesgo o asimetría positiva.

D'Agostino y Pearson [D'Agostino 73] [D'Agostino 90] presentaron un estadístico que combina los dos anteriores para dar un test omnibus de normalidad, esto es:

$$K^2 = Z^2\left(\sqrt{b_1}\right) + Z^2\left(\sqrt{b_2}\right) \tag{3.11}$$

donde $Z\left(\sqrt{b_1}\right)$ y $Z(b_2)$ son las aproximaciones normales a $\sqrt{b_1}$ y b_2.

El estadístico K^2 tiene aproximadamente una distribución chi-cuadrado con dos grados de libertad cuando la población se distribuye como una normal. Los tests basados en este estadístico son significativos para valores pequeños de K^2.

Estadístico de Shapiro-Wilk:

Uno de los tests de normalidad más populares está basado en el estadístico de Shapiro-Wilk W [Shapiro 65].

$$W = \frac{\sum_{i=1}^{n} \left(a_{i,n} X_{(i)}\right)^2}{\sum_{i=1}^{n} \left(X_{(i)} - \overline{X}\right)^2} \tag{3.12}$$

donde $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ son los estadísticos de orden de la muestra y los valores de los pesos $a_{i,n}$ están disponibles en tablas.

Los tests basados en este estadístico son significativos para valores pequeños de W. Hay que decir también que en su versión original, el test de Shapiro-Wilk sólo podía aplicarse a muestras pequeñas. Royston [Royston 82], extendió el test para poder tratar con muestras de tamaño hasta dos mil.

Otro estadístico relacionado es el de Shapiro-Francia W' [Shapiro 72]. En ambos casos se trata de tests ómnibus que no indican la forma en la que se difiere de la normalidad. Más recientemente Brown y Hettmansperger han desarrollado estadísticos más informativos en la línea de los anteriores [Brown 96].

Estadístico Z_p:

Este estadístico se debe a Lin y Mudholkar [Lin 80] y está basado en la independencia característica de la media y la varianza muestral de una muestra normal. Utiliza la transformada de Fisher Z_p del coeficiente de correlación de Pearson r entre X_i y las raices cúbicas de Wilson-Hilferty Y_i de las varianzas de la muestra excluyendo X_i. Más específicamente, el estadístico se define como:

$$Z_p = \tanh^{-1} r \tag{3.13}$$

donde $r = corr\left(X_i, Y_i\right)$ y $Y_i = \left\{ \frac{\sum_{j \neq i}^{n} X_j{}^2 - \left(\sum_{j \neq i}^{n} X_j\right)^2}{n-1} \right\}^{1/3}$

y la hipótesis nula se rechaza para valores grandes de $\left|Z_p\right|$.

Familia de estadísticos $K_{m,n}$:

La familia de estadísticos propuesta por Vasicek [Vasicek 76], está motivada por una caracterización de entropía máxima de la normalidad. Los estimadores no paramétricos de entropía están dados por:

$$K_{m,n} = \frac{n}{2ms}\left\{\prod_{i=1}^{n}\left(X_{(i+m)} - X_{(i-m)}\right)\right\}^{1/n} \tag{3.14}$$

donde $X_{(i)}$ son los estadísticos de orden de la muestra con $X_{(i)} = X_{(1)}$ para $i < 1$ y $X_{(i)} = X_{(n)}$ para $i > n$,

$$s^2 = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}{n},$$

y m es un entero cuyos valores óptimos se discuten en el artículo de Vasicek. La hipótesis de normalidad es rechazada para valores pequeños de $K_{m,n}$.

Finalmente, reseñar que existen muchos otros estadísticos que se utilizan habitualmente en tests de normalidad. Nuestra selección no pretende ser exhaustiva sino representativa de diferentes formas de detectar la no normalidad. En cualquier caso, como se verá posteriormente, el trabajo desarrollado es totalmente independiente de la elección particular de unos determinados estadísticos.

3.3 Enfoque Bayesiano del Problema

Como ya se ha visto, en el enfoque bayesiano de un problema de contraste de hipótesis, se hace imprescindible especificar de forma explícita cualquier modelo alternativo a la hipótesis nula H_0. Por este motivo, los contrastes de significación, a los que nos referimos en la sección anterior, no tienen un equivalente bayesiano, al quedar indeterminada la hipótesis alternativa. Hay que decir también, que esta obligación de explicitar alternativas, es considerada por algunos investigadores como una debilidad de la metodología bayesiana. Si se consideran alternativas explícitas no incluidas en H_0, es necesario evaluar de alguna manera sus méritos relativos a ésta, antes de decidir si se rechazan o no. Para evaluar estos méritos, el contexto bayesiano sí que proporciona

unos buenos indicadores del grado de soporte que los datos dan a las hipótesis. Estas indicadores son las probabilidades a posteriori. Aunque en algunos casos particulares (usualmente tests de una cola y si se utilizan *priors* impropios), los valores-P pueden ser numéricamente iguales a las probabilidades a posteriori, en general, esto no va a ser así. Las ventajas que se derivan del cálculo de las probabilidades a posteriori son múltiples y se expondrán más adelante.

3.3.1 Selección de Alternativas para el Problema de la Normalidad: el Sistema de Distribuciones de Johnson

Hemos hablado de la necesidad de explicitar alternativas al modelo que queremos verificar. Si este modelo es la distribución normal, podría parecer, en principio, un objetivo demasiado ambicioso tratar de considerar todas las distribuciones no normales, lo que obviamente es imposible. Lo que ocurre es que, en la práctica, aunque las distribuciones monodimensionales de datos reales observados difieren notablemente, es posible agrupar muchas de ellas (las unimodales) en cuatro tipos generales [Alan 94]:

- Distribuciones simétricas con un solo máximo.
- Distribuciones asimétricas con un solo máximo.
- Distribuciones extremadamente sesgadas o distribuciones con forma de J.
- Distribuciones con forma de U.

Lo que se ha hecho es elegir como alternativa una familia de distribuciones lo suficientemente flexible como para que estén contemplados estos cuatro tipos generales y además exista un procedimiento sencillo que permita obtener muestras de cualquier distribución de la familia, de tal manera que puedan ser utilizadas en las simulaciones que se realicen. El sistema de distribuciones de Johnson [Johnson 49], reune estos dos requisitos fundamentales.

Este sistema se basa en encontrar una función simple $g(x)$, expresable en términos de unos pocos parámetros, de tal manera que si la variable observada x sigue un amplio rango de distribuciones, $g(x)$ se distribuya de forma aproximadamente normal. Se consideran transformaciones del tipo:

$$\xi = \gamma + \delta g\left(\frac{x-\mu}{\lambda}\right) \tag{3.15}$$

donde μ, λ, γ y δ son parámetros y g la función monótona convenientemente elegida. Como puede deducirse de la expresión, μ y γ son parámetros de posición, mientras que λ y δ aparecen como parámetros de escala y se suponen positivos.

Para lograr cubrir una extensa gama de distribuciones sin que la forma de $g(\ )$ se complique excesivamente, se hace necesario distinguir diferentes subsistemas S_L, S_B y S_U, que corresponden a diferentes expresiones de $g(\ )$. En la figura 3.1 se representan estos subsistemas en términos de β_1 y β_2 [Alan 94].

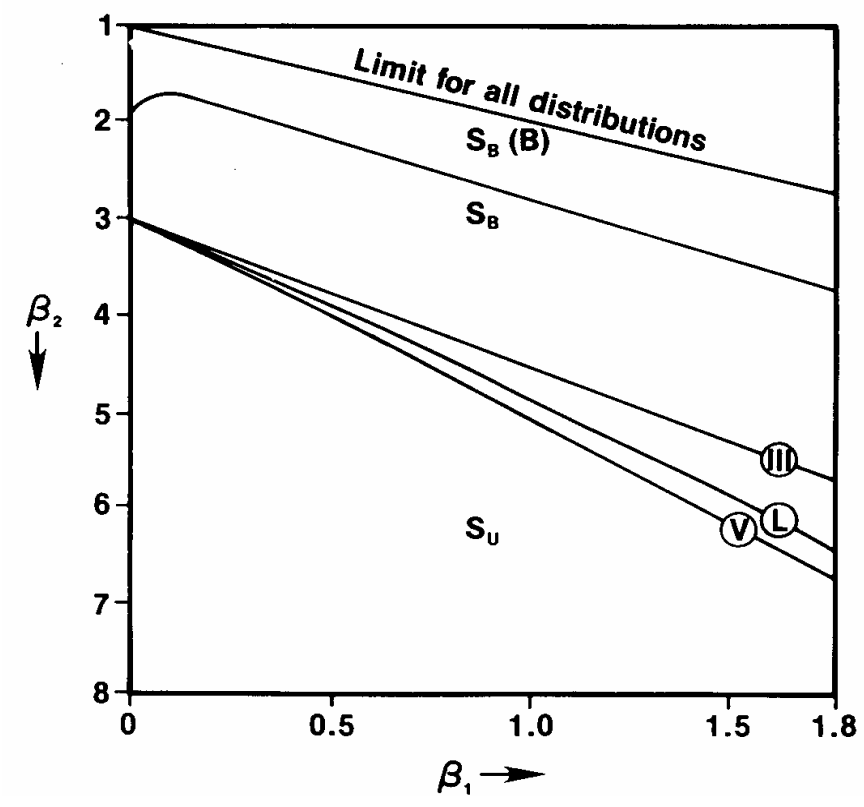

Figura 3.1. Representación esquemática del sistema de distribuciones de Johnson en β_1 y β_2 (figura extraída de [Alan 94])

Cada punto en este cuadro representa una distribución de parámetros β_1 y β_2 diferentes. La recta que marca el límite para cualquier distribución es la solución a la

ecuación $\beta_2 - \beta_1 - 1 = 0$. La recta marcada con una L representa al sistema S_L y como puede observarse define la curva que separa los otros dos subsistemas.

Pasemos a describir cada uno de los subsistemas con más detalle. Sin pérdida de generalidad, podemos escribir:

$$y = \frac{x - \mu}{\lambda} \tag{3.16}$$

Subsistema S_L o sistema lognormal: $g(y) = \log y$.

Tenemos entonces que $\xi = \gamma + \delta \log y$ y la función densidad es:

$$p(y) = \frac{\delta}{y\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(\gamma + \delta \log y)^2\right\}, \quad 0 \le y < \infty \tag{3.17}$$

Se trata de distribuciones unimodales en las que el sesgo es siempre positivo y además son leptokúrticas. En la figura 3.2 se muestra un ejemplo de distribución perteneciente a este subsistema.

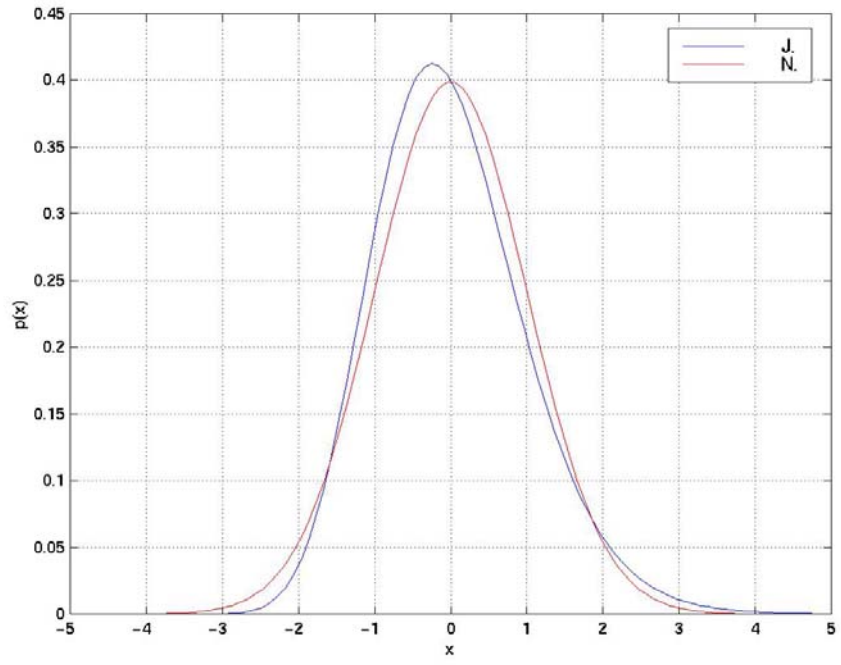

Figura 3.2. Función densidad correspondiente a la distribución de Johnson (J) de parámetros (0.5,3.45), con la distribución normal (N) superpuesta

Subsistema S_B: $g(y) = \log\{y/(1-y)\}$.

Tenemos ahora que $\xi = \gamma + 2\delta tanh^{-1}(2y-1)$ y la función densidad viene dada por:

$$p(y) = \frac{\delta}{\sqrt{2\pi}} \frac{1}{y(1-y)} \exp\left\{-\frac{1}{2}\left(\gamma + \delta \log \frac{y}{1-y}\right)^2\right\}, \quad 0 \le y \le 1 \qquad (3.18)$$

Ahora las distribuciones pueden ser bimodales en forma de U. Los momentos de las distribuciones de este subsistema son muy difíciles de determinar de forma más o menos sencilla. En la figura 3.3 se muestran algunos ejemplos representativos del subsistema S_B.

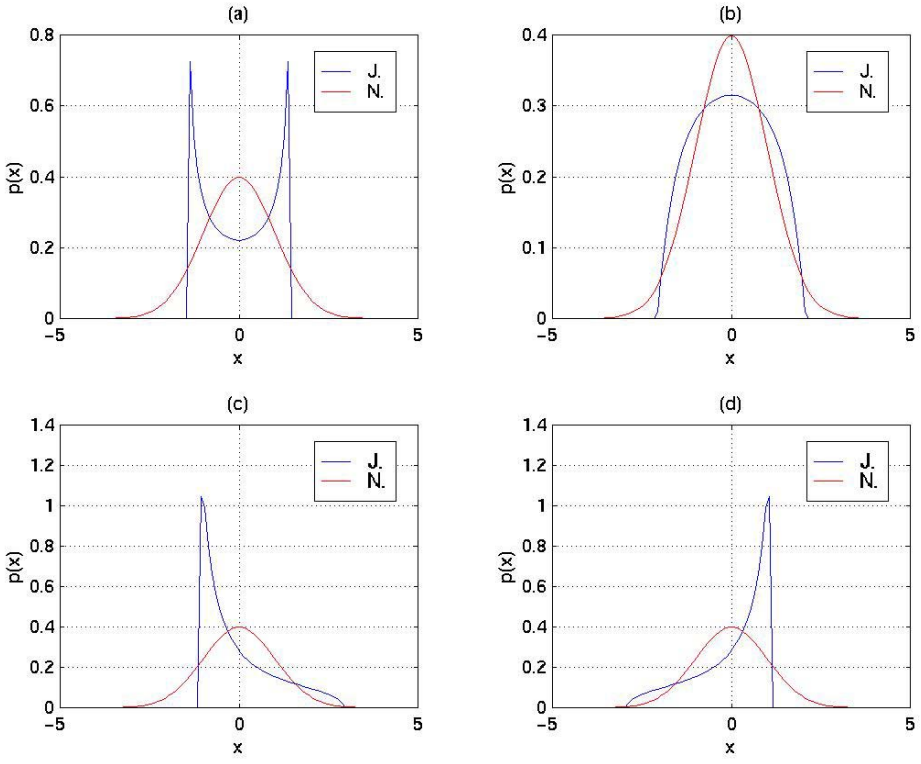

Figura 3.3. Funciones densidad correspondientes a las distribuciones de Johnson (J) de parámetros: (a) (0,1.5), (b) (0,2), (c) (1,3) y (d) (-1,3), con la distribución normal (N) superpuesta

Subsistema S_u: $g(y) = sinh^{-1} y = \log\left\{y + \sqrt{y^2+1}\right\}$.

En este caso $\xi = \gamma + \delta sinh^{-1} y$ y la función densidad es:

$$p(y) = \frac{\delta}{\sqrt{2\pi}} \frac{1}{\sqrt{y^2+1}} \exp\left[-\frac{1}{2}\left[\gamma + \delta \log\left\{ y + \sqrt{y^2+1} \right\} \right]^2 \right], \quad -\infty < y < \infty \qquad (3.19)$$

En este sistema, las distribuciones son sólo unimodales y pueden tener sesgo tanto positivo como negativo como se muestra en la figura 3.4.

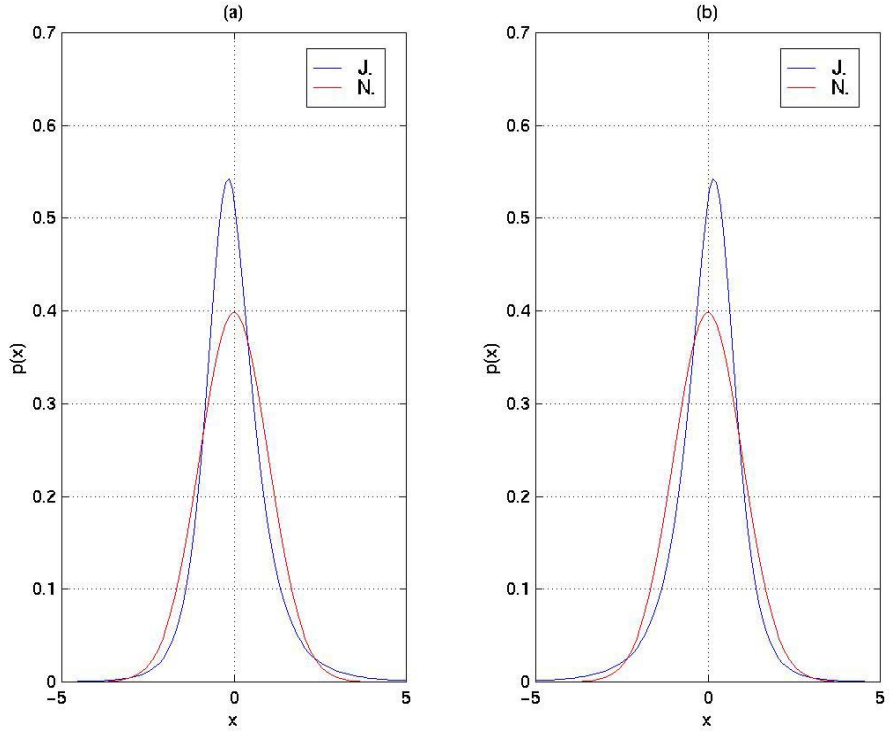

Figura 3.4. Funciones densidad correspondientes a las distribuciones de Johnson (J) de parámetros: (a) (1,10) y (b) (-1,10), con la distribución normal (N) superpuesta

3.4 Aproximación al Problema Usando Redes Neuronales

En esta sección se presenta el primero de los enfoques alternativos con los que se ha abordado el problema de la verificación de la normalidad. Básicamente, se han planteado una serie de experimentos con los que se pretende poner de manifiesto la eficiencia de la red neuronal como sistema combinador de estadísticos, tratando de mejorar el comportamiento de cada uno de ellos por separado.

Todos los experimentos se realizaron con ayuda del paquete de simulación MATLAB [MATLAB 96]. Esta es una herramienta muy utilizada en el ámbito científico-técnológico por su potencia y versatilidad, especialmente en el manejo de

estructuras de datos como vectores y matrices. Se ha hecho especial uso de los complementos *Toolbox* de redes neuronales [Toolbox-Neur. 96] y estadística [Toolbox-Stats. 96]. En el apéndice se incluye el código correspondiente a algunas de las funciones implementadas.

3.4.1 Reformulación del Problema

Para la discusión que sigue, vamos a reformular el problema de la verificación de la normalidad en una terminología más propia de la clasificación de patrones. Así, el objeto a clasificar será el conjunto de datos observados $X_1, X_2, \ldots, X_N$ y se considerarán dos clases bien definidas. Por un lado, una clase C_0 que denominaremos clase de normales y a la cual pertenecerá cualquier muestra de tamaño N que provenga de una distribución normal modimensional de media y varianza arbitrarias; y por otro lado, una clase C_1, de no normales, que agrupará a todas aquellas muestras de tamaño N, de media y varianza cualesquiera, que provengan de cualquier distribución no normal. Como ya se ha comentado en la parte de fundamentos, el vector de características se compone de medidas de ciertas propiedades de los objetos a clasificar. En nuestro caso, estas medidas vendrán dadas por los valores de los estadísticos mencionados en la sección 3.2.3. Con todo esto, la idea será construir un test que intente minimizar los criterios de error ya vistos en la sección 3.1. Puesto en los nuevos términos, tenemos un problema de clasificación convencional con algunos matices importantes que conviene destacar.

Aunque en la práctica, la gran mayoría de las veces, el número de datos de los que se dispone para entrenar un clasificador es muy limitado, en este caso no sufrimos de tal limitación. Efectivamente, podemos disponer de todas los datos que se requieran en el entrenamiento, sin más que generar artificialmente nuevas muestras. En nuestro planteamiento, el problema surge en la generación de patrones de entrenamiento para la clase C_1, ya que, (por la propia definición de C_1) es imposible abarcar todas las posibilidades y se hace necesario restringir el rango de distribuciones de trabajo (distribuciones de Johnson). Si bien, será importante comprobar el buen comportamiento del clasificador no sólo en el rango de distribuciones del entrenamiento, sino en una situación más real en la que los datos podrán venir de

cualquier distribución no normal. En una sección posterior se explica detalladamente como se han preparado los patrones de entrenamiento.

3.4.2 Elección del Clasificador: Red Neuronal

Sea $f_D\left(y^i;w\right)$ la salida de un sistema discriminante (ver capítulo de fundamentos) de parámetros w para el patrón de entrada y^i, y consideremos una función de error cuadrática del tipo:

$$E=\frac{1}{2}\sum_{i=1}^{N}\left\|f_D\left(y^i;w\right)-f_T^i\right\|^2 \tag{3.20}$$

donde f_T^i representa el valor deseado para la salida del sistema cuando la entrada es el patrón y^i y la suma se realiza sobre N patrones. Es un resultado bien conocido [Hampshire 90], que es posible ajustar los parámetros w para minimizar la función de error anterior, de tal manera que las salidas del discriminante puedan ser interpretadas como las probabilidades a posteriori bayesianas de pertenencia a las clases.

Para que esto sea factible, es necesario que se cumplan determinadas condiciones fundamentales que pasamos a comentar detenidamente.

- En primer lugar, el conjunto de datos de entrenamiento debe ser lo suficientemente grande para aproximar $N\to\infty$. En nuestro caso, esta condición puede cumplirse ya que, como se comentó anteriormente, a diferencia de lo que suele ocurrir habitualmente, podemos disponer de todos los patrones de entrenamiento necesarios sin mas que generar muestras de tamaño N de cualquiera de las distribuciones implicadas.
- Un segundo requerimiento se refiere a la capacidad del discriminante para implementar una función tan general y compleja como exija la propia dificultad del problema, en este caso la aproximación de las probabilidades a posteriori. Para conseguir esto, el número de parámetros adaptivos también deberá ser lo suficientemente grande.
- En tercer lugar, el procedimiento utilizado en la optimización de los parámetros del sistema deberá ser tal que permita alcanzar el mínimo apropiado de la función de coste.

Estas condiciones pueden satisfacerse con diferentes tipos de sistemas discriminantes o clasificadores que en general serán no lineales y de entre todos ellos hemos optado por las redes neuronales. La razón principal de esta elección es su flexibilidad a la hora de aproximar cualquier *mapping* arbitrario no lineal, multidimensional.

Como se trata de un problema de clasificación, cada vector de entrada en el conjunto de entrenamiento vendrá etiquetado de acuerdo con la clase a la que pertenece y esto quedará representado por el conjunto de valores f_T^i. Existen diferentes formas de codificar esta pertenencia. Una de las más convenientes es el esquema de codificación 1 de c, en el cual, para el vector de entrada y^i perteneciente a la clase C_l, se toma $f_T^i = \delta_{kl}$, donde δ_{kl} es el símbolo de la delta de Kronecker, tal que $\delta_{ij} = 1$ si $i = j$, y 0 de lo contrario.

Por otro lado, las ventajas que conlleva el poder estimar las probabilidades a posteriori son múltiples [Bishop 95]:

- La primera y la más importante es poder disponer de una medida condicionada a los datos que sirva como indicador del grado de apoyo a la hipótesis de normalidad, a diferencia de lo que ocurre con los valores-P, como ya se explicó en una sección anterior.
- Asignando la muestra a clasificar a la clase con probabilidad a posteriori mayor, se garantiza la minimización del error promedio de clasificación.
- Si las probabilidades a priori para un problema cambian respecto a las que se han utilizado en el entrenamiento, es muy sencillo calcular las nuevas probabilidades a posteriori sin tener que reentrenar la red. Para ello, no hay más que dividir las salidas de la red por las probabilidades a priori correspondientes al conjunto de entrenamiento (pueden estimarse a partir de la fracción de patrones en cada clase), multiplicarlas después por las nuevas probabilidades a priori y finalmente normalizar los resultados.
- Por último, el poder interpretar las salidas como probabilidades a posteriori, nos permite fijar un umbral, por debajo del cual se considere que es muy dudoso poder clasificar la muestra como normal o no normal. Por ejemplo, si la probabilidad a

posteriori está en el rango 0.5 ± 0.1. De esta manera se rechazarían algunos casos a los que se podría aplicar otro procedimiento que resultara más fiable.

3.4.3 Preparación de las Pruebas Realizadas

En los diferentes apartados que componen esta sección, se describe como se han preparado las pruebas realizadas, especialmente en aquello que tiene que ver con el entrenamiento de las redes neuronales.

3.4.3.1 Tipo de Red Neuronal Utilizada

Como modelo de red neuronal hemos elegido el perceptrón multicapa con una capa oculta. El número de neuronas en la capa de entrada será igual al número de estadísticos que utilicemos y el número de neuronas de la capa de salida será igual al número de clases, en este caso dos. El que no está fijado a priori es el número de neuronas en la capa oculta ya que depende del problema particular a tratar. En adelante y mientras no se especifique lo contrario, se utilizarán cinco neuronas, ya que después de realizar numerosas simulaciones con diferentes patrones de entrenamiento (como se explica en el apartado siguiente), hemos encontrado, en general, resultados satisfactorios.

3.4.3.2 Preparación del Conjunto de Patrones de Entrenamiento

Los estadísticos que se utilizaron como entradas son: β_1, β_2, W, Z_p, $K_{3,n}$ y $K_{5,n}$. Para determinar los patrones que configuran el conjunto de entrenamiento, se generaron muestras de diferentes tamaños y se calcularon los valores de los estadísticos sobre estas muestras. Los tamaños de muestra considerados fueron $N = 25, 50, 100, 200$. Llegados a este punto, hay dos cuestiones fundamentales a plantearse: cómo generar las muestras y cuántas generar.

Respecto a la primera cuestión, el generar muestras de la distribución normal no suponía ningún problema. Sin embargo, en lo que respecta a la clase de no normales, representada por las distribuciones de Johnson, si que se plantean algunas dificultades. Hay que tener en cuenta que el sistema de Johnson lo constituyen un continuo no acotado de distribuciones, parametrizado en β_1 y β_2. Por lo tanto, de entrada, se hacía necesario acotar el rango de distribuciones a considerar. Así, se impuso la restricción

$0 \le \beta_1 \le 1$ y $1 \le \beta_2 \le 10$ y aunque las cotas resulten un tanto arbitrarias, son lo suficientemente grandes para cubrir una amplia gama de alternativas a la normal. Además para soslayar el hecho de trabajar con un rango continuo de valores de los parámetros, se procedió a discretizar el problema. El procedimiento de discretización es muy simple y consiste en hacer un sorteo uniforme bidimensional dentro de los límites previamente establecidos del plano sesgo-kurtosis. De esta manera se obtienen unos puntos al azar, donde cada punto representa una distribución con sesgo y kurtosis diferentes, resultando ahora muy sencillo generar una o más muestras de cada punto. Cabe pensar que si el número de puntos es lo suficientemente grande podremos aproximarnos al continuo todo lo que queramos. En adelante y a menos que se especifique lo contrario, trabajaremos siempre con mil puntos (mil distribuciones), ya que hemos constatado que es una cantidad suficiente y al aumentarla no hemos observado ningún cambio significativo en los resultados encontrados.

La segunda cuestión importante tenía que ver con el número de muestras a generar o, en definitiva, el número de patrones con los que se entrenará la red. Después de numerosas pruebas, hemos comprobado que en el peor de los casos, esto es, muestras de tamaño veinticinco (mayor varianza en los estadísticos) y mayor número de entradas (considerando todos los estadísticos al mismo tiempo), cincuenta mil patrones resultaban suficientes y garantizaban dos cifras decimales significativas en los resultados encontrados. Por este motivo, en adelante y a menos que se especifique lo contrario trabajaremos con este número de patrones.

Otra variable del problema que viene determinada por el conjunto de entrenamiento son las probabilidades a priori de las clases $P(C_0)$ y $P(C_1)$, ya que estas probabilidades pueden estimarse a partir de las proporciones en las que intervienen los patrones en el conjunto de entrenamiento. Así, de los cincuenta mil patrones, veinticinco mil serán normales y los otros veinticinco mil no normales, dando unas probabilidades a priori de $1/2$ para cada clase. Esta elección parece la más razonable si no se tiene un conocimiento a priori que permita suponer otra cosa. Queda pendiente aclarar en que proporción se generaron los patrones dentro de la clase de las distribuciones alternativas. Igualmente optamos por considerar el mismo número de patrones en cada punto. Así, para obtener los veinticinco mil patrones de Johnson, se generaron veinticinco muestras de cada distribución. En terminología bayesiana, esto

equivale a utilizar *priors* no informativos [Berger 85], lo que en el caso continuo general puede resultar extremadamente complicado pero que se simplifica notablemente en un caso discreto como este.

Finalmente, otra forma de conocimiento a priori viene dada por cuestiones de invarianza. Efectivamente, a priori se sabe que cualquier procedimiento diseñado para la discriminación entre la normalidad y la no normalidad, debería ser invariante ante cambios en la media o varianza de los datos a la entrada. Existen diversas formas de incorporar este conocimiento a priori [Bishop 95]. En este caso, como los propios estadísticos son aproximadamente invariantes, para un tamaño de muestra dado, la red neuronal respetará la invarianza requerida.

Antes de pasar a otro apartado, señalar que aunque, a efectos de claridad, hemos fijado el número de neuronas, de distribuciones en el plano sesgo-kurtosis y de patrones de entrenamiento, hemos comprobado, dependiendo del caso particular, que es posible disminuir significativamente estos números y seguir encontrando buenos resultados.

3.4.3.3 Algoritmos de Entrenamiento de la Red

Una vez explicada la preparación del conjunto de datos para el entrenamiento, pasamos a comentar los algoritmos utilizados para la optimización de los pesos de la red. Se probaron tres algoritmos: el clásico backpropagation, backpropagation con momento y ritmo de aprendizaje adaptivo y el de Levenberg-Marquardt [Levenberg 44], [Marquardt 63]. Aunque con todos se conseguía el aprendizaje de la red, este último, sin duda, ha dado los mejores resultados desde el principio y por eso es el que se ha utilizado en todas las pruebas. El único inconveniente que presenta es la carga computacional asociada, ya que demanda bastante memoria, pero en cualquier caso no ha supuesto un gran problema.

En lo que se refiere a la inicialización de los pesos, se ha seguido el método propuesto por Nguyen y Widrow [Nguyen 90], y hemos comprobado que, efectivamente, acelera de forma notable el aprendizaje, respecto a una inicialización uniforme.

3.4.3.4 Preparación del Conjunto de Patrones de Validación

Para la estimación de los errores de clasificación, atendiendo a los diferentes criterios, utilizaremos otro conjunto de patrones diferente al de entrenamiento pero obtenido de la misma manera. Si bien en un problema habitual de clasificación es importante disponer de dos conjuntos diferentes para entrenar y validar, a fin de evitar predicciones demasiado optimistas, en este caso, realmente viene a ser igual debido a la gran cantidad de patrones con los que se trabaja.

3.4.4 Prueba 1: Demostración de que la Red Neuronal es Capaz de Estimar las Probabilidades a Posteriori de las Clases

Como ya se ha comentado, si se cumplen determinadas condiciones, las salidas de una red neuronal pueden ser interpretadas como las probabilidades a posteriori de las clases condicionadas a las entradas. Se han realizado una serie de experimentos que pretenden demostrar la validez de esta afirmación para nuestro problema particular.

A modo de ilustración, se escogieron los estadísticos $\sqrt{\beta_1}$ y β_2 como entradas. Lo primero que se hizo fue buscar un procedimiento, alternativo a las redes neuronales, que permitiera calcular con bastante precisión las probabilidades a posteriori. Para ello, se estimaron las densidades de probabilidad de estos estadísticos condicionadas a las clases $p\left(\left(\sqrt{\beta_1}, \beta_2\right) | C_i\right)$ y luego se aplicó la regla de Bayes. Para estimar las densidades se utilizó el método clásico no paramétrico, basado en funciones kernel, al que se hizo referencia en la parte de fundamentos. En este caso se optó por un kernel normal y los parámetros h_i de la ventana fueron fijados de forma automática por las rutinas de Matlab utilizadas. En la estimación se utilizaron un millón de muestras normales y un millón de muestras de Johnson de tamaño cincuenta. En estas condiciones se obtuvieron las densidades que se muestran en la figura 3.5.

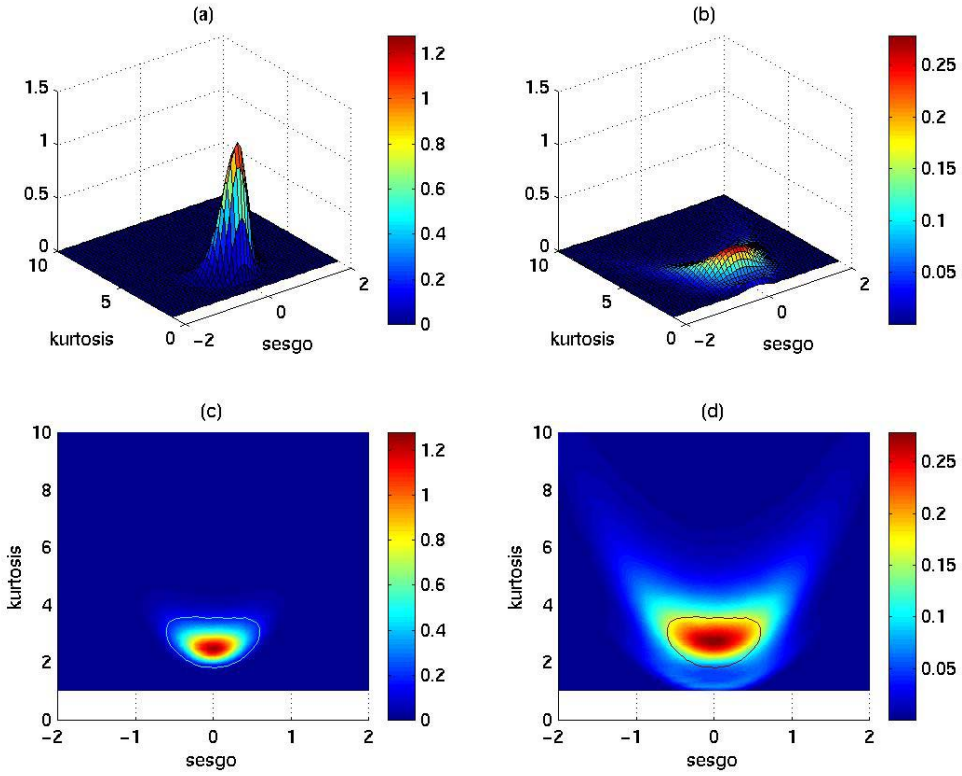

Figura 3.5. Densidades de probabilidad correspondientes a $p\left(\left(\sqrt{\beta_1}, \beta_2\right) | C_0\right)$ (a), (c), y a $p\left(\left(\sqrt{\beta_1}, \beta_2\right) | C_1\right)$ (b), (d). (c)y (d) son las mismas gráficas que (a) y (d), pero vistas desde arriba

Si tomamos lo anterior como referencia válida en la aproximación a las probabilidades a posteriori, será interesante compararlo con las salidas de una red neuronal entrenada con un número mucho menor de patrones (veinticinco mil normales y veinticinco mil no normales). En la figura 3.6 se comparan las probabilidades a posteriori de la clase de las normales $P\left(C_0 \mid \left(\sqrt{\beta_1}, \beta_2\right)\right)$, obtenidas a partir de las densidades estimadas y los resultados encontrados para dos redes con tres y diez neuronas en la capa oculta.

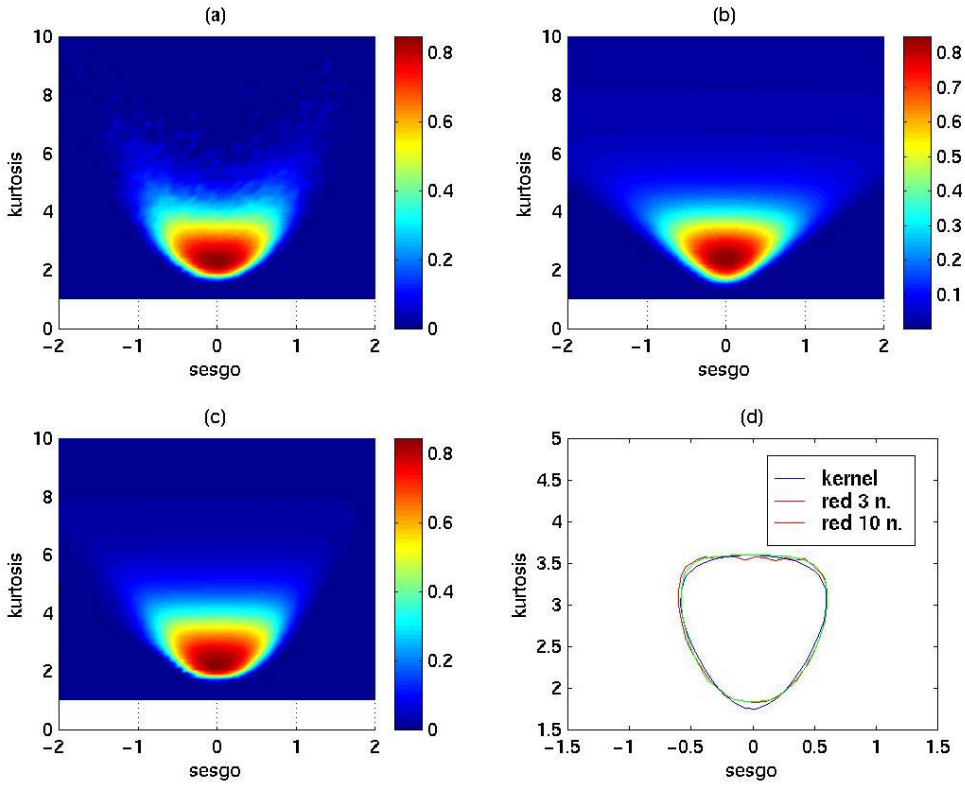

Figura 3.6. Visualización bidimensional de las probabilidades a posteriori $P\left(C_0 \mid \left(\sqrt{\beta_1}, \beta_2\right)\right)$ para (a) la aproximación usando funciones kernel, (b) red neuronal con 3 neuronas, y (c) red neuronal con 10 neuronas. En (d) se muestran los contornos de probabilidad 0.5 para los tres casos

Se puede observar que la red entrenada con diez neuronas aproxima mejor la forma de la probabilidad a posteriori, aunque a efectos prácticos la frontera de decisión sea prácticamente la misma en los tres casos.

En la figura 3.7 se muestra de nuevo la probabilidad a posteriori calculada con la red de diez neuronas pero representada en una escala ampliada. Es muy importante observar que, para valores de sesgo y kurtosis no presentes en los patrones de entrenamiento, el comportamiento de la red es el óptimo ya que, como es deseable, les asigna valores de probabilidad muy cercanos a cero. Hemos comprobado que este buen comportamiento es el habitual, a menos que se empleen muchas neuronas, en cuyo caso podrían aparecer efectos "extraños" en las regiones donde no se ha entrenado la red, debido a su excesiva flexibilidad.

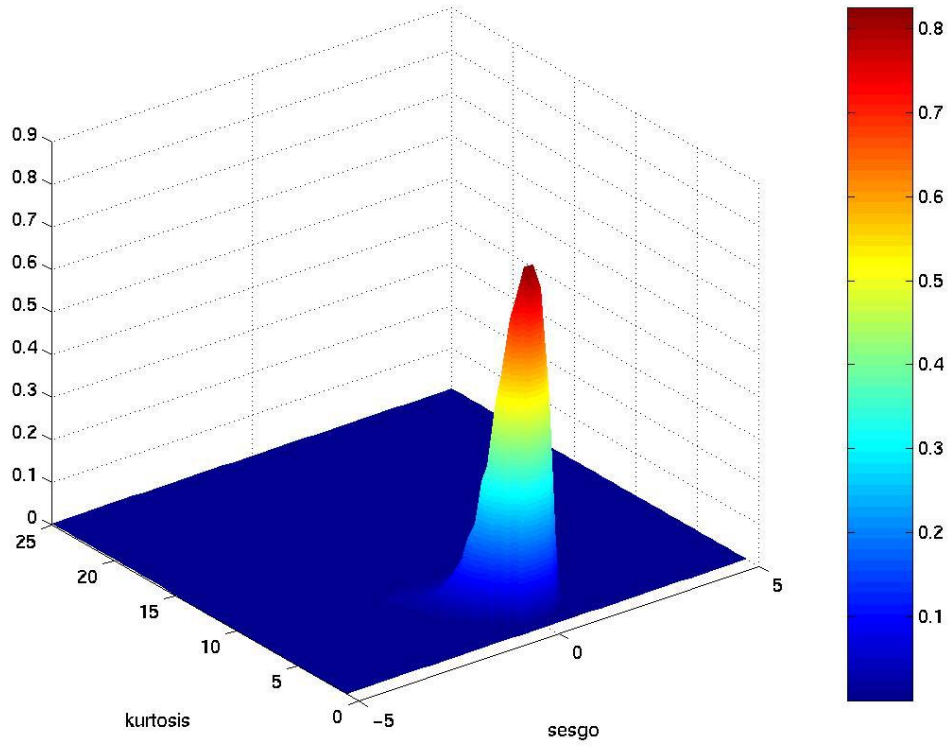

Figura 3.7. Probabilidad a posteriori con la red de 10 neuronas en una escala ampliada en sesgo y kurtosis

Finalmente, comentar que también se ha utilizado otro indicador de la bondad de la red para aproximar las probabilidades a posteriori, siendo además muy fácil de calcular.

Para ello, se tuvo en cuenta el hecho de que el promedio de cada una de las salidas de la red sobre todos los patrones en el conjunto de entrenamiento, debería aproximar las probabilidades a priori de las clases [Bishop 95], ya que:

$$P(C_k) = \int P(C_k \mid y) p(y) dy \cong \frac{1}{N} \sum_{i=1}^{N} P(C_k \mid y^i) \tag{3.21}$$

Si se comparan estos *priors* estimados con las fracciones de patrones correspondientes a cada clase, dentro del conjunto de entrenamiento, nos puede servir como indicación del grado de bondad de las estimaciones de las probabilidades a posteriori. De hecho, con este procedimiento tan sencillo y rápido, hemos encontrado en general un altísimo grado de acuerdo, lo cual nos ha servido de también de orientación para valorar lo adecuado del procedimiento.

3.4.5 Prueba 2: La Red como Sistema para Combinar Estadísticos. Criterio de Bondad: Probabilidad Promedio de Error

En la sección 3.2.3 se describieron algunos estadísticos usados con frecuencia en los tests de normalidad. Existen numerosas publicaciones en las que se recoge la eficiencia de tests basados en estos estadísticos, normalmente expresada en forma de tablas con las potencias de rechazo a unas pocas distribuciones arbitrarias diferentes de la normal.

Cabe plantearse la posibilidad de combinar estos estadísticos de alguna manera con el fin de mejorar la eficiencia de cada uno de ellos por separado. En ocasiones esta combinación puede ser muy simple y limitarse a una forma cuadrática definida positiva como ya se ha visto en el caso del estadístico de D'Agostino y Pearson $K^2 = Z^2\left(\sqrt{b_1}\right) + Z^2\left(\sqrt{b_2}\right)$. Parece entonces que la red neuronal podría ser una buena opción para combinar diferentes estadísticos, sin más que añadirlos como nuevas entradas. Con el fin de evaluar lo eficaz que resulta esta combinación, se han realizado una serie de experimentos con diferentes criterios de error, empezando por el error promedio.

En el contexto bayesiano en el que nos movemos, el criterio más natural para medir la bondad del clasificador es el error promedio. Como ya se indicó en la parte de fundamentos, este error puede expresarse como:

$$E = P(C_0)E_0 + P(C_1)E_1 \tag{3.22}$$

donde en este caso, E_0 es el error tipo I, es decir, el error que se comete al clasificar muestras normales como no normales y E_1 es el error tipo II que nos da cuenta de la otra posibilidad de equivocarnos, esto es, clasificar como normales, muestras no normales.

En la tabla 3.1 se muestran los errores promedio de clasificación obtenidos para redes neuronales entrenadas con cada uno de los estadísticos por separado y con la combinación de todos ellos (para los diferentes tamaños de muestra).

N	β_1	β_2	W	Z_p	$K_{3,n}$	$K_{5,n}$	$(\beta_1, \beta_2, W, Z_p, K_{3,n}, K_{5,n})$
25	0.3766	0.3726	0.3545	0.3748	0.4073	0.4181	0.3335
50	0.3152	0.3022	0.2729	0.3211	0.3556	0.3696	0.2415
100	0.2517	0.2241	0.1868	0.2666	0.2819	0.2861	0.1636
200	0.1914	0.1526	0.1002	0.2131	0.2066	0.1948	0.0854

Tabla 3.1

Se observa, que como era de esperar, en todos los casos hay una disminución del error importante con respecto a algunos estadísticos, aunque no demasiado significativa respecto al estadístico de Shapiro-Wilk, corroborando la buena reputación que éste tiene como indicador de las desviaciones de la normalidad. Es importante también volver a insistir en el hecho de que, si como ya se ha comprobado, las salidas de las redes neuronales son capaces de aproximar aceptablemente las probabilidades a posteriori, tendríamos la mejor clasificación posible con la información que aportan estos estadísticos o características.

Los estadísticos utilizados en el entrenamiento de las redes neuronales llevan a procedimientos tipo ómnibus, esto es, sensibles a desviaciones de la normalidad por causas variadas. En ocasiones, puede resultar más informativo diseñar un test que sea especialmente sensible a aspectos concretos de la no normalidad. De entre ellos, las desviaciones por asimetría o sesgo diferente de cero, tienen especial interés práctico, ya que uno de los rasgos a destacar de la forma de la distribución normal es su simetría. Existen diferentes estadísticos que permiten caracterizar la simetría o asimetría de una distribución monodimensional [Arnold 95].

Llamando μ, m, y M a la media, mediana y moda respectivamente, tenemos:

- $\gamma_1 = \dfrac{E(X-\mu)^3}{\sigma^3} = \sqrt{\beta_1}$, que ya había sido comentado en una sección anterior.
- $\gamma_m^1 = \dfrac{(\mu - m)}{\sigma}$, estadístico atribuido a Yule (1912)
- $b = \dfrac{(Q_3 + Q_1 - 2m)}{Q_3 - Q_1}$, donde Q_1 y Q_3 representan el primer y tercer cuartiles de la distribución. Atribuido a Bowley (1920).

- $S_k = \frac{(\mu - M)}{\sigma}$, aparentemente introducido por Karl Pearson (1895), tiene la dificultad de la determinación de la moda. Sin embargo, para una clase amplia de distribuciones conocida como distribuciones de Pearson, y de las cuales hablaremos posteriormente, puede expresarse exactamente en términos de los primeros cuatro momentos, quedando en la forma:
- $S_k = \frac{\sqrt{\beta_1}(\beta_2 + 3)}{2(5\beta_2 - 6\beta_1 - 9)}$, y de esta manera se define una medida de asimetría aplicable a todas las distribuciones con μ_4 (momento de orden 4) finito.

Combinando estos estadísticos con una red neuronal podemos diseñar un test de sesgo o asimetría. En la tabla 3.2 se muestran los valores de los errores promedio encontrados como resultado de dicha combinación comparándolos con las redes entrenadas con cada uno de los estadísticos por separado.

N	$\sqrt{\beta_1}$	γ_m^1	b	S_k	$\left(\sqrt{\beta_1}, \gamma_m^1, b, S_k\right)$
25	0.3763	0.4478	0.4733	0.4248	0.3630
50	0.3155	0.4223	0.4646	0.3682	0.2841
100	0.2517	0.3895	0.4535	0.3116	0.2012
200	0.1917	0.3421	0.4293	0.2566	0.1046

Tabla 3.2

En este caso, sobre todo para tamaños de muestra 100 y 200, si que se observa una disminución significativa general del error promedio, dejando constancia de la mejora en la eficiencia respecto a cada uno de los estadísticos por separado.

Decir, por último, que el conocimiento del tipo de desviación de la normalidad, sirve, entre otras cosas, como orientación en un procedimiento para intentar hacer la muestra normal a través de algún tipo de transformación.

3.4.6 Prueba 3: La Red como Sistema para Combinar Estadísticos. Criterio de Bondad: Maximización de la Potencia

Debido a la imposibilidad de definir un error promedio, con el fin de evaluar la bondad de un test en el contexto clásico, se suele utilizar el criterio de fijar el error tipo I a un valor arbitrario y tratar de minimizar el tipo II, o lo que es lo mismo, maximizar la potencia para diferentes distribuciones alternativas. En lo que se refiere a los valores a los que fijaremos los errores tipo I, se utizarán los tres niveles de referencia habituales: 0.01, 0.05 y 0.1. Para calcular la potencia se han elegido tres conjuntos de distribuciones alternativas a la normal:

Conjunto 1: el primer conjunto lo constituyen los puntos de una rejilla regular en el plano sesgo-kurtosis del sistema de Johnson dada por:

$$\sqrt{\beta_1} = \{0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1\}$$
$$\beta_2 = [1,1.25,1.5,1.75,2,2.25,2.5,2.75,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10]$$

Conjunto 2: El segundo conjunto lo constituyen distribuciones con los mismos valores de sesgo y kurtosis pero tomadas de otro sistema muy conocido, el sistema de distribuciones de Pearson [Pearson 63].

Al igual que el sistema de Johnson, el de Pearson trata de reunir en una familia los cuatro casos generales de distribuciones mencionados en la sección 3.3.1. Se puede demostrar que las distribuciones solución de la ecuación:

$$\frac{df}{dx} = \frac{(x-a)f}{b_0 + b_1 x + b_2 x^2} \tag{3.23}$$

corresponden a dichos casos.

En la figura 3.8 se muestra el sistema de distribuciones de Pearson en función de β_1 y β_2. Se observa que aparte del límite superior para todas las distribuciones hay también un límite inferior que delimita aquellas distribuciones para las que el momento de orden ocho no existe.

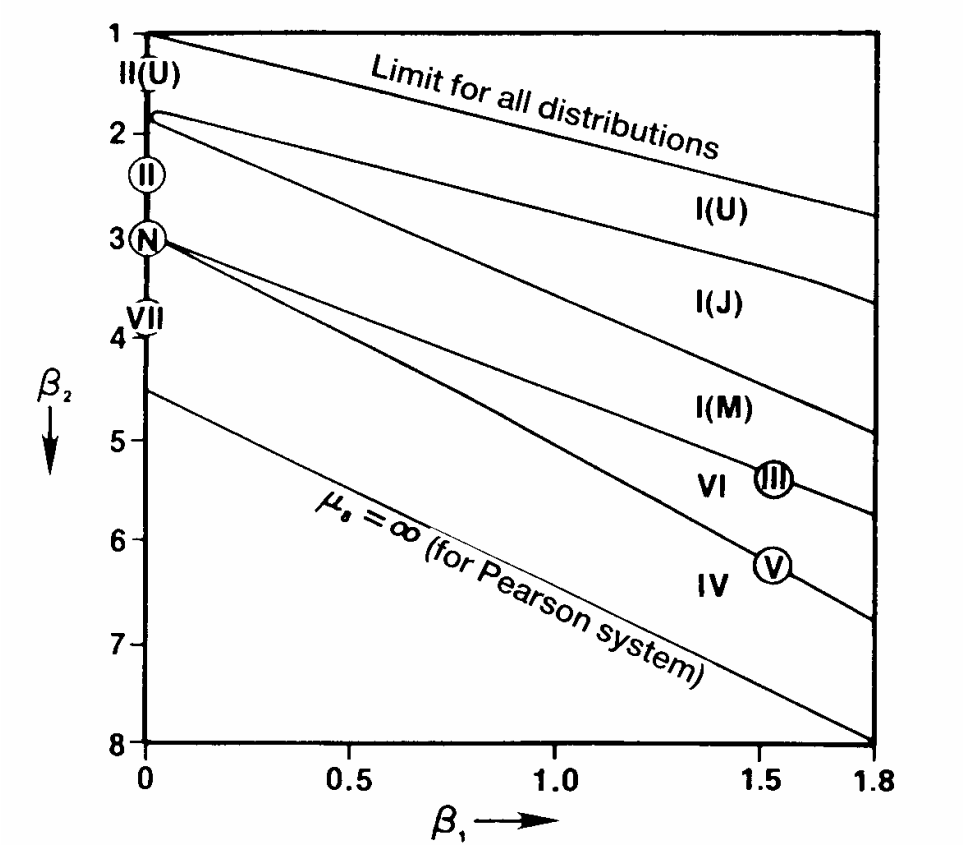

Figura 3.8. Representación esquemática del sistema de distribuciones de Pearson en β_1 y β_2 (figura extraida de [Alan 94])

Pearson distinguió un gran número de tipos de distribuciones, algunos de interés práctico y otros totalmente triviales. Pasamos a describir brevemente algunos de los más interesantes, utilizando la notación que aparece en la figura.

Tipo I: distribución Beta de primera clase.

$$f = \frac{1}{B(p,q)} x^{p-1}(1-x)^{q-1}, \quad 0 \le x \le 1;\ p,q > 0 \tag{3.24}$$

Hay que decir que este tipo incluye tanto distribuciones unimodales I(M) como distribuciones en forma de J, I(J) y U, I(U).

Tipo II: caso simétrico del anterior.

$$f = \frac{1}{aB(m+1,m+1)}\left(1-\frac{x^2}{a^2}\right)^m, \quad -a \le x \le a \tag{3.25}$$

Tipo III: distribución Gamma

$$\text{F}=\frac{1}{\Gamma(\lambda)}x^{\lambda-1}e^{-x}, \quad \lambda > 0,\ 0 \le x < \infty \tag{3.26}$$

Tipo IV: no se le conoce con ningún nombre en particular. Hay que decir que se trata de distribuciones difíciles de manejar en la práctica, al carecer de una forma cerrada para su función de distribución.

$$f = k\left(1+\frac{x^2}{a^2}\right)^{-m} \exp\{-\nu arctan(x/a)\}, \quad m > \frac{1}{2} \tag{3.27}$$

Como una proporción importante de los puntos de nuestra rejilla regular corresponde a este tipo, se hizo necesario buscar un procedimiento para generar muestras de dichas distribuciones. El procedimiento seguido está basado en el método de rechazo (*rejection method*) [Press 92] que permite generar números aleatorios a partir de la función de densidad de probabilidad. Para la aplicación de esta técnica se precisa de una función de comparación y con este fin se eligió la función de distribución Lorentziana: $f = \frac{1}{\pi}\left(\frac{1}{1+x^2}\right)$, apropiada cuando las comparaciones se hacen con distribuciones con la forma típica de "campana".

Tipo V: tampoco se la conoce con ningún nombre particular.

$$f = \frac{\gamma^{p-1}}{\Gamma(p-1)}x^{-p}e^{-\gamma/x}, \quad 0 \le x < \infty \tag{3.28}$$

Tipo VI: distribución Beta de segunda clase.

$$f = \frac{1}{B(p,q)}\frac{x^{p-1}}{(1+x)^{p+q}}, \quad 0 \le x < \infty;\ p,q > 0 \tag{3.29}$$

Tipo VII: distribución t de Student.

$$f = \frac{1}{aB\left(\frac{1}{2}, m-\frac{1}{2}\right)}\left(1+\frac{x^2}{a^2}\right)^{-m}, \quad -\infty < x < \infty \tag{3.30}$$

Conjunto 3: el tercer conjunto lo componen cuatro distribuciones adicionales que hemos incluido por completitud ya que son muy utilizadas en la práctica. Nos referimos a la distribución uniforme, la distribución chi-cuadrado de dos grados de libertad, la distribución de Weibull con parámetro dos y una mezcla de dos distribuciones normales con medias 2 y -2 y varianza 1.

De cada una de las distribuciones de los tres conjuntos se han generado veinticinco mil muestras para el cálculo de la potencia. También se han utilizado veinticinco mil muestras normales para fijar los niveles de error antes mencionados. Como tamaños de muestra se han utilizado 25 y 100.

Las pruebas de simulación llevadas a cabo pretenden comparar la potencia de los tests clásicos con la potencia de rechazo de la red neuronal entrenada con todos los estadísticos utilizados en los tests. De hecho utilizaremos la red entrenada con los seis estadísticos como entrada y cuyos resultados se mostraron en la tabla 3.1. De nuevo, miraremos los valores de salida de la red, pero ahora, en lugar de tomar el valor 0.5 (suponiendo probabilidades a priori iguales como antes) como umbral para clasificar una muestra como normal o no normal, utilizaremos un procedimiento numérico para encontrar el valor de corte que de los errores tipo I especificados: 0.01, 0.05 y 0.1. Una vez encontrado este nuevo valor de referencia, el cálculo de la potencia resulta muy sencillo.

En la figuras 3.9, 3.10 y 3.11, se muestran los valores de potencia calculados para las distribuciones de Johnson y tamaño de muestra 25 a los niveles 0.01, 0.05 y 0.1, respectivamente.

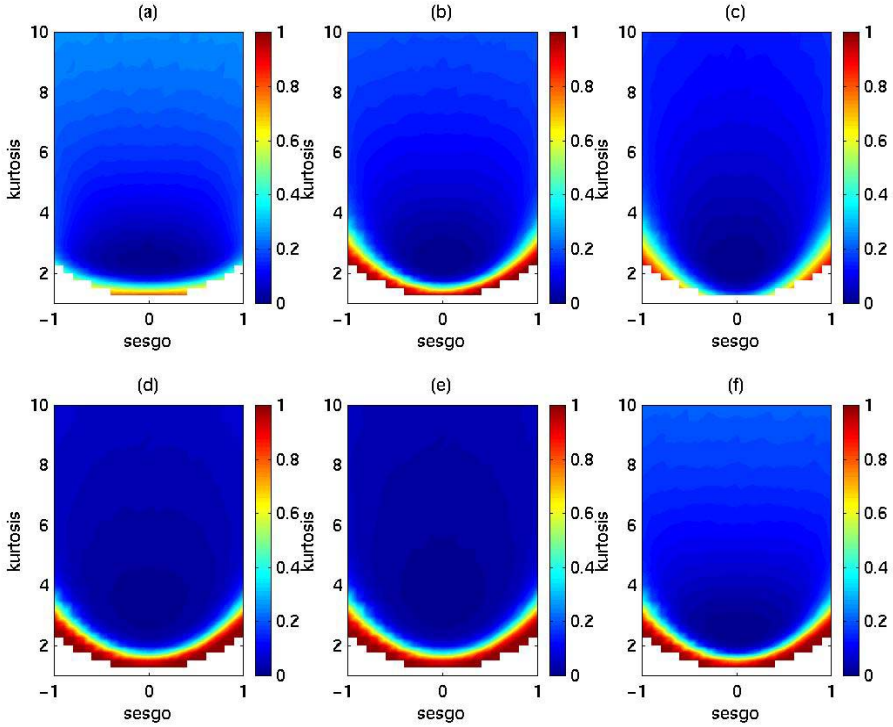

Figura 3.9. Potencias de rechazo a las distribuciones de Johnson para N=25, al nivel 0.01 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

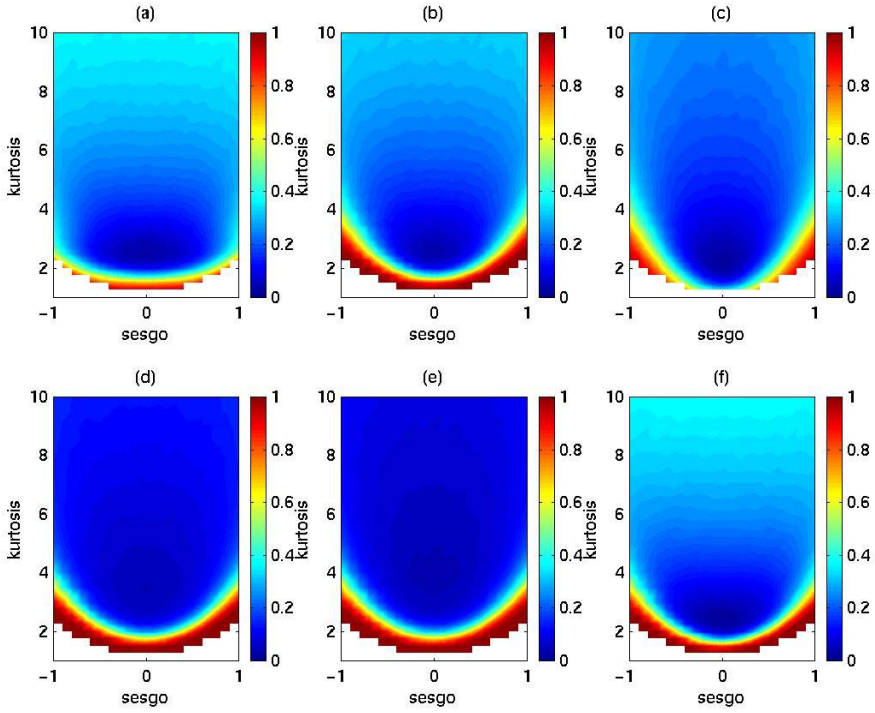

Figura 3.10. Potencias de rechazo a las distribuciones de Johnson para N=25, al nivel 0.05 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

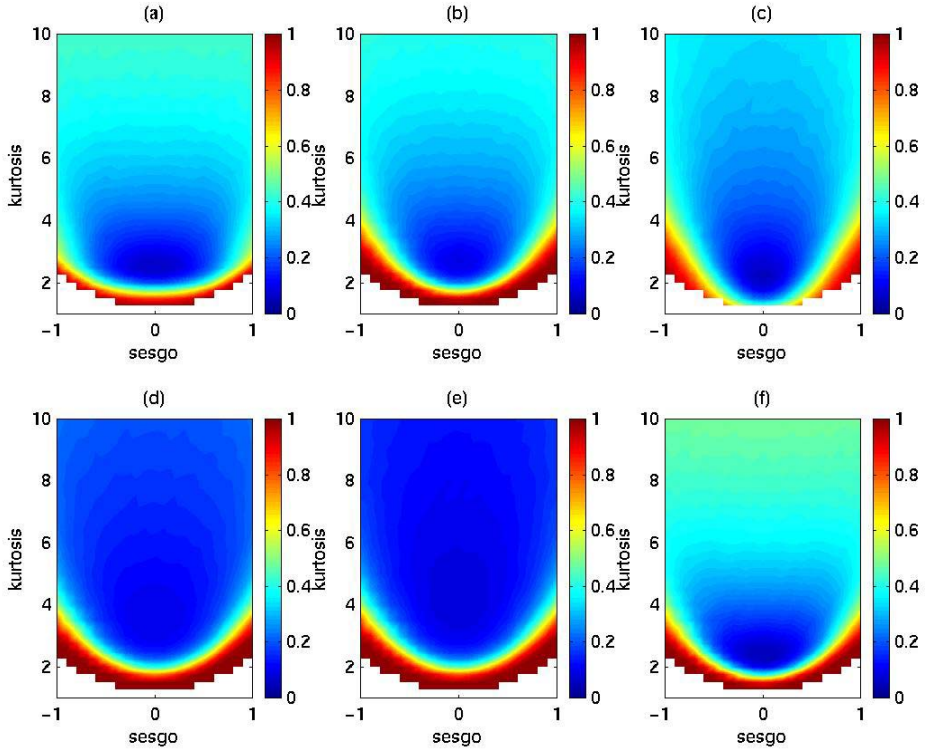

Figura 3.11. Potencias de rechazo a las distribuciones de Johnson para N=25, al nivel 0.1 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

En la figuras 3.12, 3.13 y 3.14, se muestran los valores de potencia calculados para las distribuciones de Johnson y tamaño de muestra 100 a los niveles 0.01, 0.05 y 0.1, respectivamente.

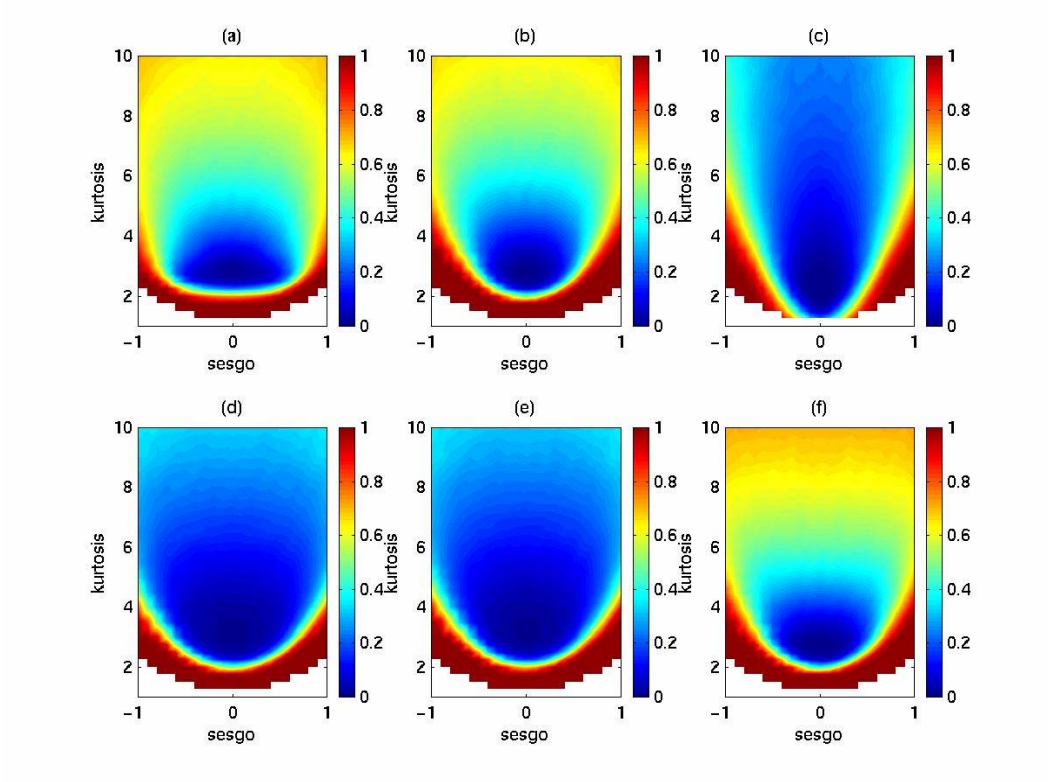

Figura 3.12. Potencias de rechazo a las distribuciones de Johnson para N=100, al nivel 0.01 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

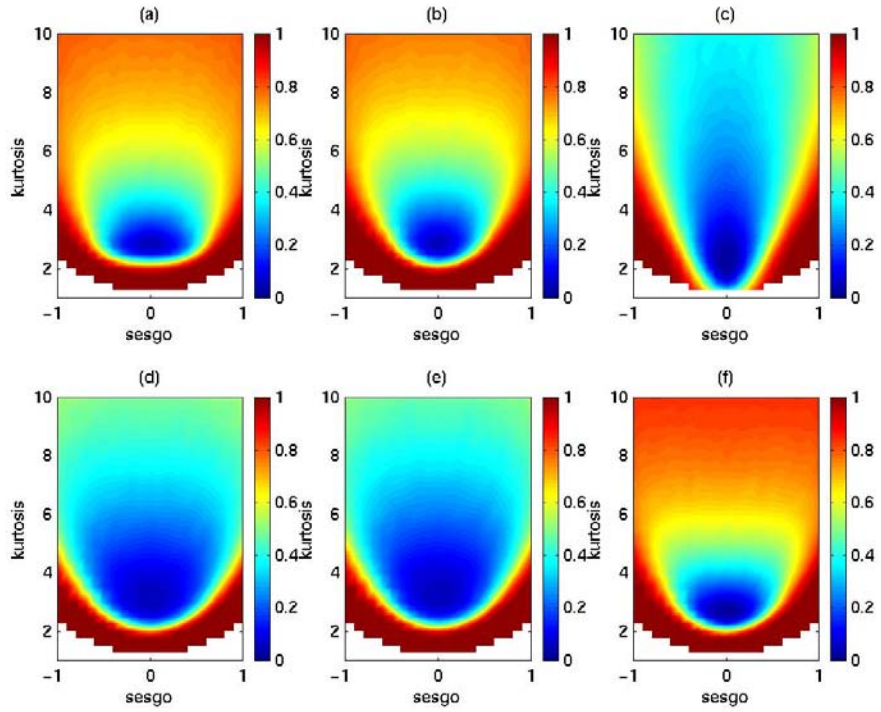

Figura 3.13. Potencias de rechazo a las distribuciones de Johnson para N=100, al nivel 0.05 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

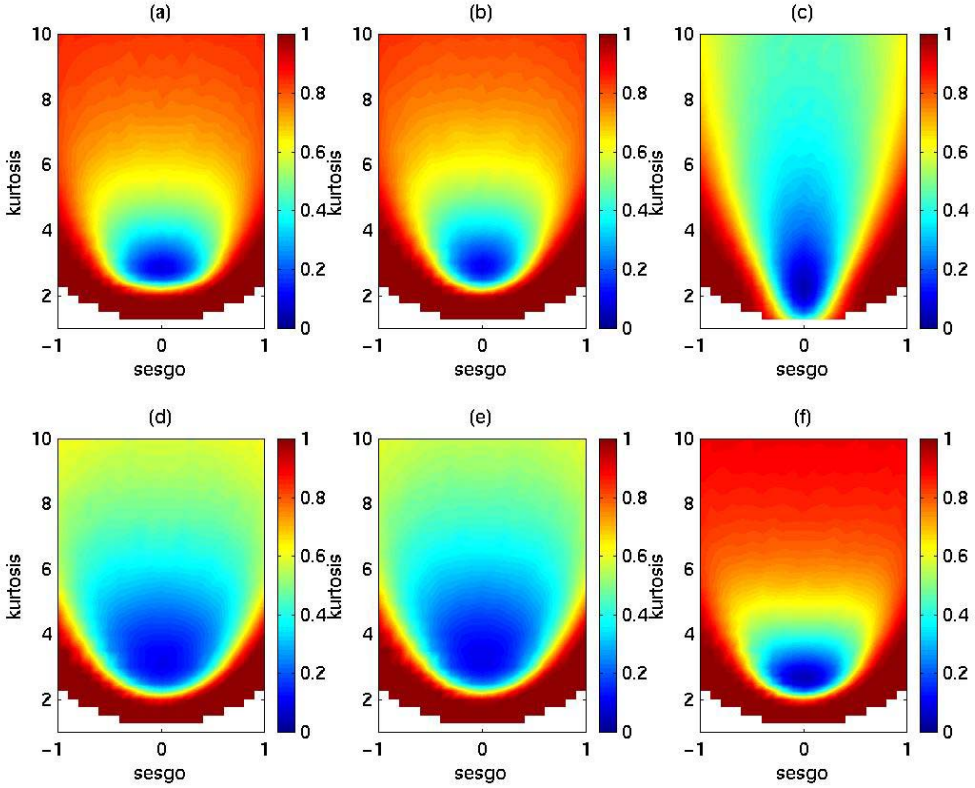

Figura 3.14. Potencias de rechazo a las distribuciones de Johnson para N=100, al nivel 0.1 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

De las figuras anteriores se pueden sacar varias conclusiones de interés. En primer lugar, sirven como ilustración de la potencia de tests de normalidad ya conocidos y permiten confirmar el comportamiento puesto de manifiesto en otras publicaciones. Así, se observa como los estadísticos K^2 y W son especialmente sensibles a las distribuciones con valores de kurtosis grandes, si bien, el estadístico de Shapiro-Wilk presenta un buen comportamiento en general. Por su parte, el estadístico Z_p sólo parece comportarse bien para distribuciones bastante asimétricas. En lo que se refiere a $K_{3,n}$ y $K_{5,n}$, se presentan como los mejores para valores de kurtosis pequeños cuando el tamaño de muestra es 25, perdiendo considerablemente si aumenta el tamaño de la muestra. En definitiva, se tiene un conjunto de estadísticos que se diferencian claramente en su capacidad de rechazo de las distribuciones dependiendo de sus valores de sesgo y kurtosis.

En lo que respecta a la red neuronal, sería deseable que, de alguna forma, recogiera las virtudes de los diferentes estadísticos con los que se entrena y en cierto modo es lo que ha ocurrido, como se puede observar en las diferentes figuras. Especialmente cuando el tamaño de la muestra es 100, la red parece presentar el mejor comportamiento general, destacando sobre todo en valores de kurtosis elevados. Para valores de kurtosis pequeños y especialmente para $N = 25$, la red se muestra menos eficiente respecto a los tests basados en $K_{3,n}$ y $K_{5,n}$.

En la figuras 3.15, 3.16 y 3.17, se muestran los valores de potencia calculados para las distribuciones de Pearson y tamaño de muestra 25 a los niveles 0.01, 0.05 y 0.1 respectivamente.

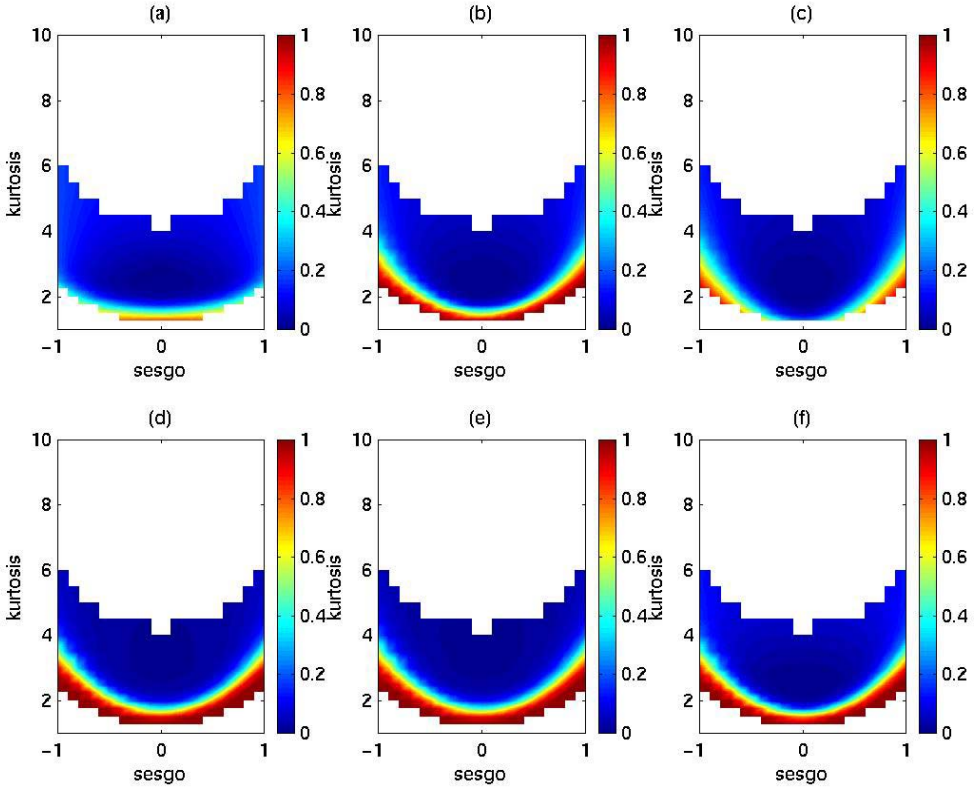

Figura 3.15. Potencias de rechazo a las distribuciones de Pearson para N=25, al nivel 0.01 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

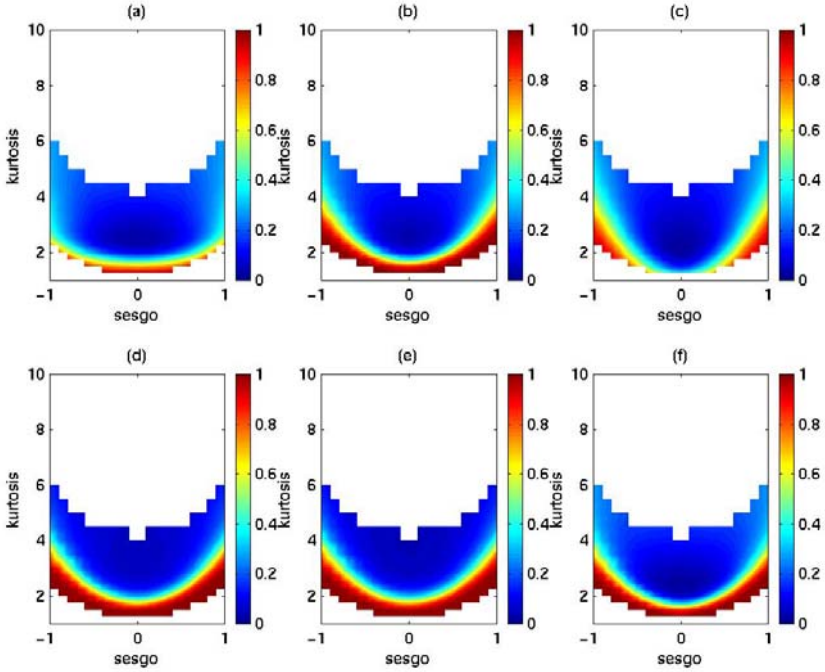

Figura 3.16. Potencias de rechazo a las distribuciones de Pearson para N=25, al nivel 0.05 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

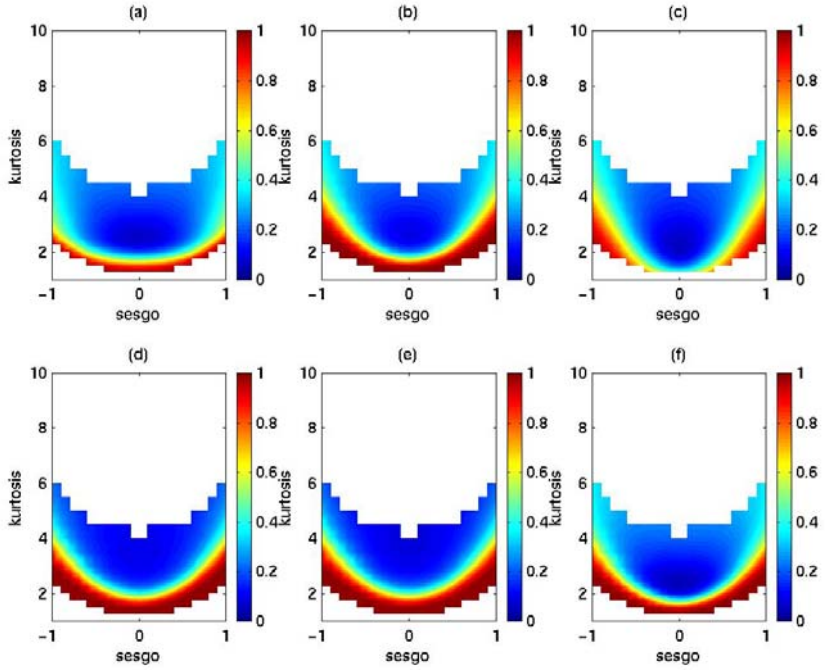

Figura 3.17. Potencias de rechazo a las distribuciones de Pearson para N=25, al nivel 0.1 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

En la figuras 3.18, 3.19 y 3.20, se muestran los valores de potencia calculados para las distribuciones de Pearson y tamaño de muestra 100 a los niveles 0.01, 0.05 y 0.1, respectivamente.

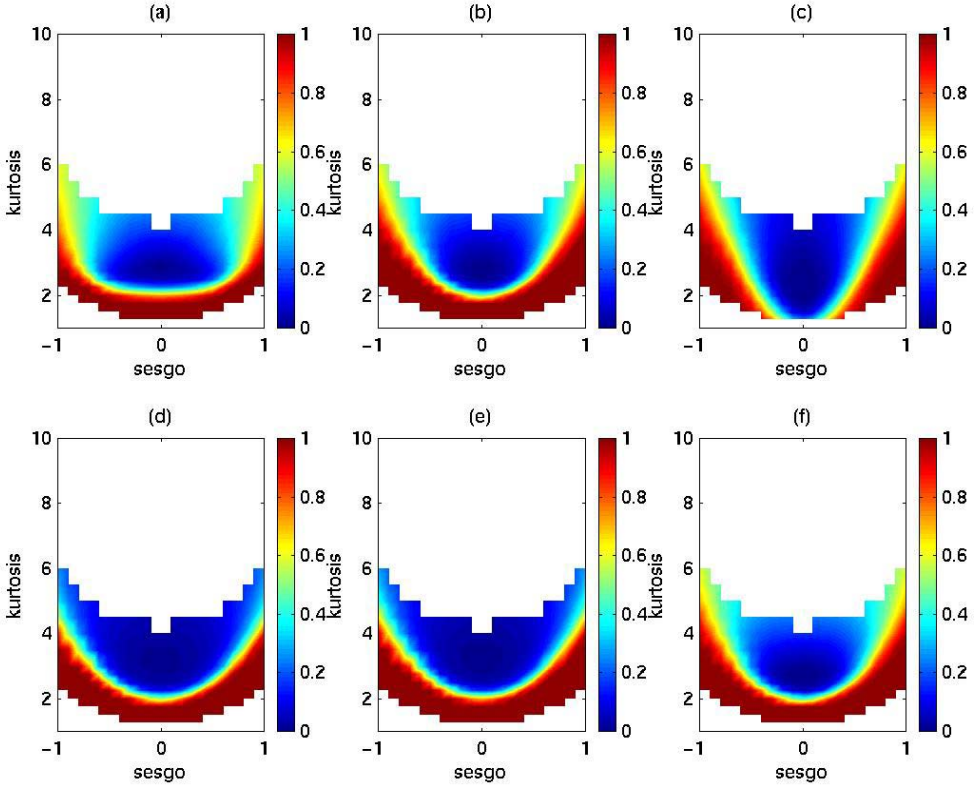

Figura 3.18. Potencias de rechazo a las distribuciones de Pearson para N=100, al nivel 0.01 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

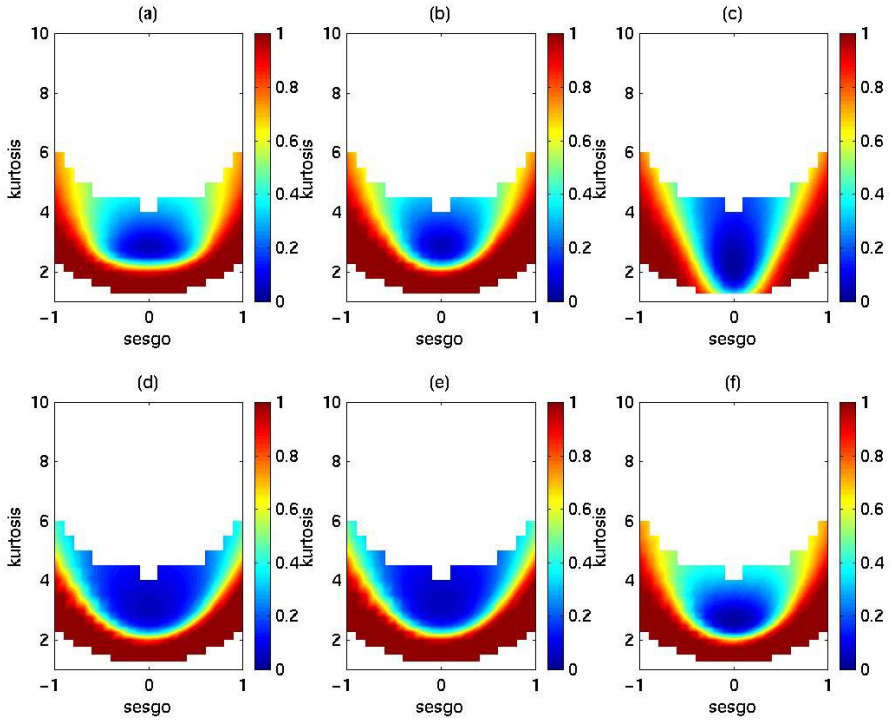

Figura 3.19. Potencias de rechazo a las distribuciones de Pearson para N=100, al nivel 0.05 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

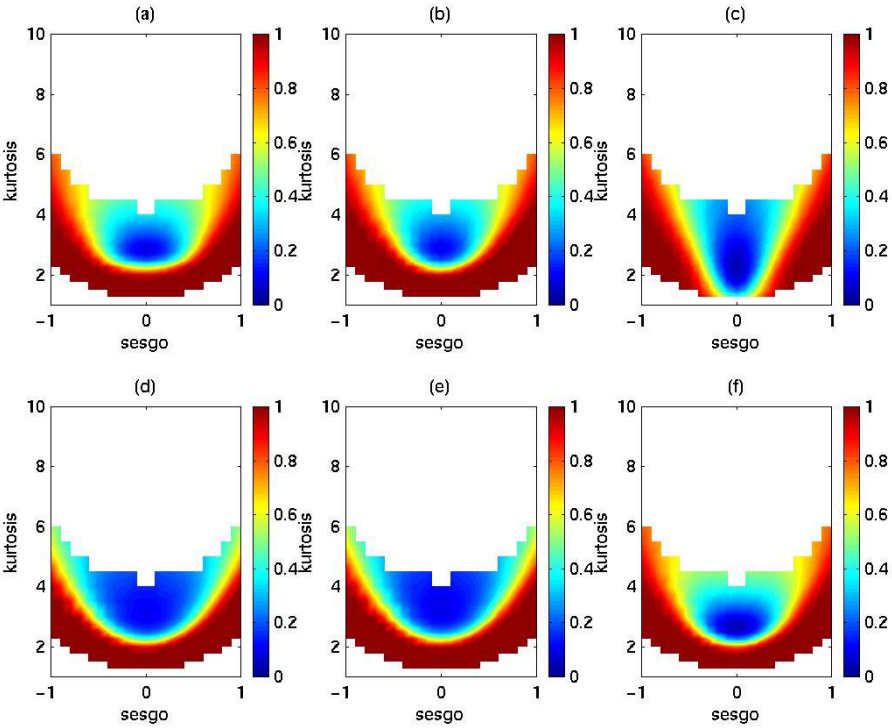

Figura 3.20. Potencias de rechazo a las distribuciones de Pearson para N=100, al nivel 0.1 de los tests basados en (a) K^2, (b) W, (c) Z_p, (d) $K_{3,25}$, (e) $K_{5,25}$ y (f) red neuronal

Aunque las distribuciones de Pearson no cubren un rango tan amplio como las de Johnson, en las zonas en las que coinciden se observa un enorme parecido en las gráficas de potencia. Si bien esto no debe sorprender demasiado, dada la similitud entre las distribuciones de los dos sistemas [Elderton 69], si que es importante porque sirve como confirmación de que la red neuronal sigue ofreciendo un buen comportamiento incluso para muestras procedentes de distribuciones diferentes a las utilizadas en el entrenamiento.

Finalmente, en la tabla 3.3 se muestran los valores de potencia encontrados para el conjunto 3 de distribuciones, con $N = 25$, al nivel 0.01, y en la 3.4, lo mismo para $N = 100$, al nivel 0.1.

	K^2	W	Z_p	$K_{3,n}$	$K_{5,n}$	$(\beta_1, \beta_2, W, Z_p, K_{3,n}, K_{5,n})$
Uniforme	0.0996	0.0548	0.0085	0.2723	0.2966	0.0955
χ^2_2	0.5479	0.7802	0.7804	0.7967	0.8152	0.7994
Weibull-2	0.0740	0.0639	0.0646	0.0465	0.0492	0.0524
Mezcla-Normales	0.4073	0.1400	0.0000	0.3262	0.3702	0.0484

Tabla 3.3

	K^2	W	Z_p	$K_{3,n}$	$K_{5,n}$	$(\beta_1, \beta_2, W, Z_p, K_{3,n}, K_{5,n})$
Uniforme	0.9845	0.9661	0.0095	0.9807	0.9958	0.9377
χ^2_2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Weibull-2	0.2995	0.5509	0.5615	0.2316	0.3372	0.4547
Mezcla-Normales	0.9997	0.9997	0.0021	0.9978	0.9991	0.9947

Tabla 3.4

De nuevo, nos encontramos con un buen comportamiento de la red en general, excepto para la mezcla de normales con $N = 25$. Sin embargo, para $N = 100$, los resultados encontrados son muy buenos, incluso en este caso.

3.4.7 Prueba 4: Red Neuronal Entrenada con Muestras de Diferentes Tamaños

Hasta ahora se ha puesto de manifiesto la capacidad de la red neuronal tanto para aproximar las probabilidades a posteriori como para combinar estadísticos mejorando así su eficiencia. En cualquiera caso, siempre que se cambie el tamaño de la muestra con la que se trabaje, nos veremos obligados a entrenar una nueva red. Sería por lo tanto interesante, plantearse si es posible que la red siga discriminando igual de bien para diferentes tamaños de muestra.

Con este fin, se ha entrenado una red neuronal utilizando como entradas los estadísticos $\sqrt{\beta_1}$ y β_2 y una entrada adicional que se utilizará para indicar el tamaño de la muestra. En el entrenamiento se han utilizado veinte mil patrones normales y veinte mil no normales. Con las mismas distribuciones que en experimentos anteriores, se han obtenido cinco mil patrones de cada uno de los tamaños de muestra con los que se ha trabajado: 25, 50, 100 y 200, haciendo el total de cuarenta mil. Cada patrón estará entonces formado por los valores de $\sqrt{\beta_1}$ y β_2 calculados sobre la muestra correspondiente y un tercer elemento que será el tamaño de dicha muestra. Hay que añadir que la red utilizada tenía seis neuronas en la capa oculta. En la figura 3.21 se muestran superpuestas las curvas de decisión, correspondientes a la probabilidad a posteriori 0.5, de esta red neuronal y diferentes redes entrenadas sólo con muestras de un tamaño.

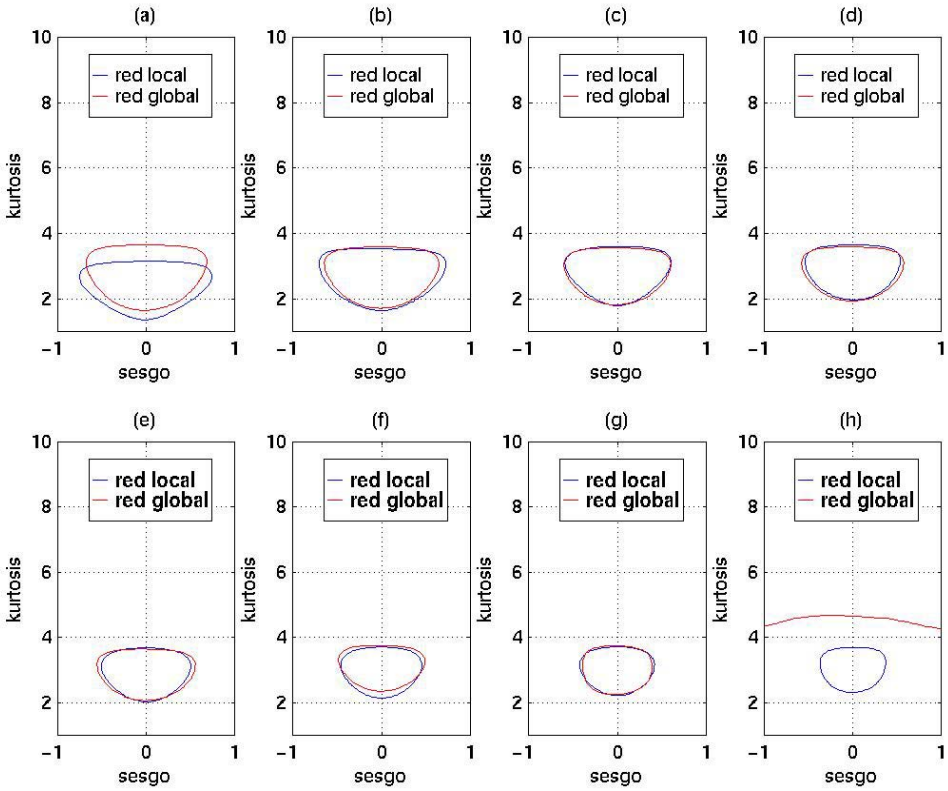

Figura 3.21. Curvas de decisión (0.5) para diferentes tamaños de muestra: (a) N=10, (b) N=25, (c) N=50, (d) N=75, (e) N=100, (f) N=150, (g) N=200, (h) N=300

Se observa que para tamaños de muestra comprendidos en el rango 25-200, la aproximación es bastante buena, incluso para 75 y 150, tamaños no presentes en el entrenamiento. Fuera de este rango, la red no aproxima adecuadamente. Los resultados encontrados son, por lo tanto, bastante razonables.

3.5 Aproximación al Problema Usando Teoría de Grandes Desviaciones

En el enfoque anterior, una de las cuestiones importantes que permitía simplificar notablemente el problema, era la discretización del espacio continuo de distribuciones de Johnson. Se asumió, sin demasiada justificación, que tomando un número lo suficientemente grande de puntos en el plano sesgo-kurtosis, la aproximación al continuo resultaba satisfactoria. Además, el procedimiento seguido para seleccionar las distribuciones alternativas se basaba puramente en el azar.

En esta segunda aproximación al problema de verificación de la normalidad de un conjunto de datos, se plantean criterios más específicos para la selección de alternativas y se utilizará Teoría de la Información [Cover 91] y Teoría de Grandes Desviaciones [Bahadur 71], [Dembo 98] para intentar justificar los resultados encontrados.

3.5.1 Planteamiento Inicial

Supongamos que como alternativa a la normal tenemos una distribución de Johnson con unos valores de sesgo y kurtosis determinados. Si además la media y la varianza también están fijadas, digamos a cero y uno respectivamente, tendríamos un problema de hipótesis simple frente a una alternativa también simple. Representa el caso más sencillo de resolver y tanto desde el punto de vista clásico como bayesiano, existe una solución óptima para el mismo. Para unas probabilidades a priori dadas $P(H_0)$ y $P(H_1)$, el criterio bayesiano que minimiza el error promedio, dará como buena la hipótesis cuya probabilidad a posteriori $P(H_i \mid x) = \dfrac{P(H_i)p(x \mid H_i)}{p(x)}$ sea mayor.

Supongamos ahora que en lugar de tener una alternativa simple, tenemos una alternativa compuesta formada por un conjunto discreto de r distribuciones de Johnson con medias cero, varianzas uno y valores de sesgo y kurtosis diferentes. El criterio bayesiano, al igual que antes, se construirá a partir del cálculo de las probabilidades a posteriori

$$P(H_0 \mid x) = \frac{P(H_0)p(x \mid H_0)}{p(x)}, \qquad P(H_1 \mid x) = \frac{P(H_1)\sum_{j=1}^{r} P(H_{1j})p(x \mid H_{1j})}{p(x)} \tag{3.31}$$

donde los $P(H_{1j})$ representan los *priors* discretos asociados a cada una de las alternativas que forman la compuesta, cumpliendose que $\sum_{j=1}^{r} P(H_{1j}) = 1$.

Evidentemente, la suposición de medias y varianzas conocidas no representa la situación práctica y por lo tanto constituye una simplicación que en el problema real hay que contemplar de alguna manera. Lo que se ha hecho es hacer un preprocesamiento de

las muestras, consistente en estandarizarlas, esto es, $\widetilde{X}_i = \dfrac{X_i - \overline{X}}{\sigma}$, donde $\overline{X} = \dfrac{1}{N}\sum_{i=1}^{N} X_i$ y $\sigma^2 = \dfrac{1}{N-1}\sum_{i=1}^{N}\left(X_i - \overline{X}\right)^2$, y de esta manera seguir con la hipótesis de medias cero y varianzas uno. Esto introduce un ruido en el procedimiento que disminuirá en la medida en que consideremos muestras más grandes.

Como aplicación de lo expuesto, consideremos el siguiente experimento en el que se utilizaron cinco mil distribuciones de Johnson con parámetros $0 \le \beta_1 \le 1$ y $1 \le \beta_2 \le 10$, obtenidos al azar con el procedimiento ya conocido. Como conjunto de datos de validación para calcular los errores promedio se utilizaron cinco mil muestras normales y cinco mil muestras de distribuciones de Johnson, una de cada distribución y para los tamaños $N = 10, 25, 50, 100, 200, 400$. Los resultados encontrados se muestran en la tabla 3.5.

N	*Errores promedio*
10	0.4248
25	0.3368
50	0.2385
100	0.1635
200	0.0874
400	0.0408

Tabla 3.5

Hay que destacar que se han añadido los tamaños de muestra 10 y 400, respecto a los empleados en el entrenamiento de las redes neuronales. Además, si se comparan estos resultados con los de la tabla 3.1, se observa que son muy similares a los de la red neuronal entrenada con todos los estadísticos. Hay que tener en cuenta también, que para valores de $N = 200$ *ó* 400, el ruido que se introduce por la normalización de las muestras no debe ser demasiado grande y por lo tanto el procedimiento estará bastante cercano al óptimo.

A continuación se realizó el mismo tipo de cálculo, seleccionando subconjuntos de diferentes números de distribuciones, del conjunto de cinco mil. En concreto, se eligieron mil subconjuntos con una, dos, tres, cuatro y cinco distribuciones alternativas. En la figura 3.22, se ilustra el proceso de selección de los subconjuntos.

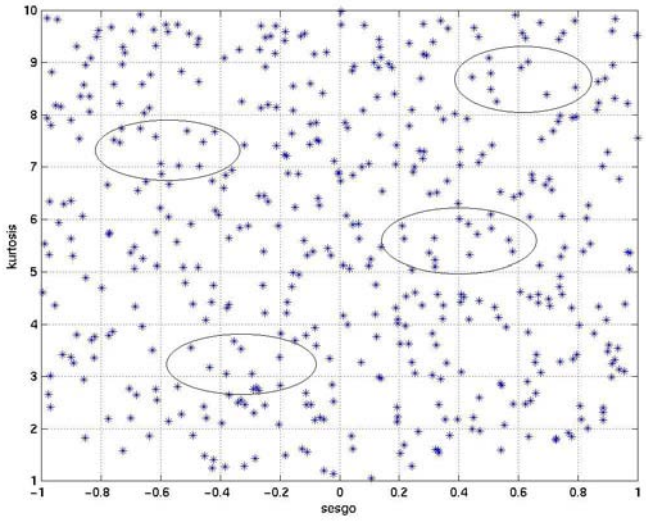

Figura 3.22. Selección de subconjuntos de distribuciones del plano sesgo-kurtosis. Cada asterisco es una distribución de sesgo y kurtosis especificados

Para calcular ahora el error promedio, se utilizó el mismo conjunto de validación que anteriormente. Como se disponía de una cantidad suficiente de información, en cada uno de los casos, se hizo una estimación de la densidad de probabilidad como se muestra en la figura 3.23.

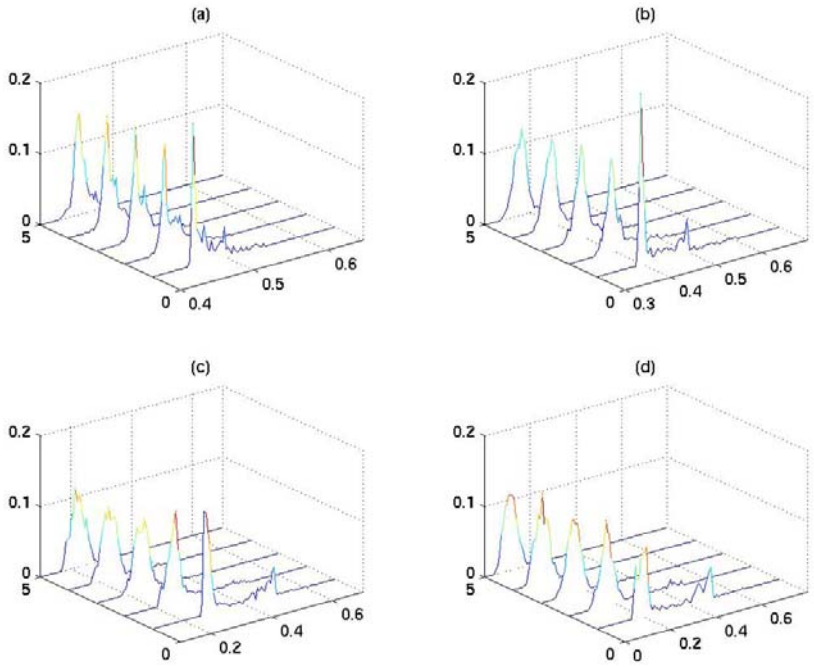

Figura 3.23. Estimaciones de las densidades de probabilidad de los errores promedio calculados con subconjuntos de diferente número de distribuciones alternativas, para (a) N=10, (b) N=25, (c) N=100 y (d) N=400

Para ayudar a la interpretación de la figura, se incluye también la tabla 3.6 con algunos estadísticos medidos sobre estos datos para $N = 400$.

N° puntos	*Media*	*Varianza*	*Mínimo*	*Máximo*
1	0.2794	0.0149	0.1605	0.6378
2	0.1900	0.0061	0.0913	0.4989
3	0.1500	0.0018	0.0543	0.4464
4	0.1288	0.0012	0.0519	0.3964
5	0.1140	0.0008	0.0497	0.3009

Tabla 3.6

De la figura y tabla anteriores pueden extraerse interesantes conclusiones que pasamos a comentar. En primer lugar y como es lógico, en todos los casos se observa que a medida que se consideran subconjuntos con un número mayor de distribuciones, la media va disminuyendo para parecerse cada vez más a los valores que aparecen en la tabla 3.5, y

la varianza también va disminuyendo progresivamente. Si nos fijamos en los valores mínimos nos encontramos una evolución similar pero de una manera un tanto sorprendente. Se observa por ejemplo, que para $N = 400$, es posible escoger solamente tres distribuciones como alternativa y obtener un error promedio muy similar al calculado con los cinco mil puntos. Si bien para $N = 10$ no se detectan cambios significativos al ir aumentando el número de puntos, en el resto de los casos se da un fenómeno similar al que se produce para tamaño 400.

Se tiene entonces que parece factible seleccionar un número muy pequeño de distribuciones de Johnson y diseñar un test que resulte prácticamente equivalente al diseñado con una buena aproximación al continuo (cinco mil distribuciones). También es cierto, como queda patente en los valores máximos registrados, que no toda selección es igualmente buena. En la tabla 3.7, se muestran los puntos seleccionados (sesgo y kurtosis) correspondientes a los mínimos para los diferentes tamaños de muestra (subconjuntos de tres distribuciones). Decir que hemos excluido el caso $N = 10$ por considerarlo excesivamente "ruidoso".

N	*Punto* 1	*Punto* 2	*Punto* 3
25	$(-0.4516,3.0012)$	$(-0.1319,5.4882)$	$(0.8274,4.4497)$
50	$(-0.5784,2.6903)$	$(-0.1306,4.6242)$	$(0.6484,4.4286)$
100	$(-0.5784,2.6903)$	$(-0.1306,4.6242)$	$(0.6484,4.4286)$
200	$(-0.4516,3.0012)$	$(-0.1319,5.4882)$	$(0.8274,4.4497)$
400	$(-0.4516,3.0012)$	$(-0.1319,5.4882)$	$(0.8274,4.4497)$

Tabla 3.7

Es interesante destacar la gran coincidencia existente entre los tamaños 25, 200 y 400 por un lado y 50 y 100 por el otro, teniendo en cuenta que la selección se ha hecho entre mil subconjuntos.

Para tratar de justificar estas coincidencias en los resultados encontrados, se ha recurrido a la aproximación asintótica dentro del contexto de la Teoría de Grandes Desviaciones y Teoría de la Información. A continuación se exponen algunas de las ideas fundamentales que se aplicarán posteriormente a este problema particular.

3.5.2 Aproximación Asintótica y Teoría de Grandes Desviaciones

En pocas palabras, se podría definir la Teoría de Grandes Desviaciones como una teoría acerca de sucesos que ocurren con muy poca frecuencia. En la actualidad, constituye uno de los campos más activos de la teoría de la probabilidad con muchas y sorprendentes ramificaciones. Una de sus aplicaciones es al análisis de las colas de distribuciones de probabilidad y como veremos a continuación, éste es de hecho el aspecto que más nos interesa para nuestro problema. Así, utilizaremos algunos teoremas fundamentales como el Teorema de Chernoff que nos servirá de base para justificar muchos de los resultados encontrados.

Si hasta ahora nos habíamos planteado diseñar tests óptimos para muestras de tamaño finito N, sería interesante extender esta discusión a muestras con $N \to \infty$. Para ello, se hace necesario redefinir algunos de los conceptos utilizados con anterioridad. De entrada, va a resultar conveniente expresar los errores I y II de otra forma, totalmente equivalente a las ya vistas.

Error I:

$$P_0\left(\frac{1}{N}\sum_{i=1}^{N}\log\frac{p(X_i \mid H_1)}{p(X_i \mid H_0)} \geq \frac{1}{N}\log\frac{P(H_0)}{P(H_1)}\right) \tag{3.32}$$

donde $P_0(\ )$ representa la probabilidad de que ocurra el suceso entre paréntesis supuesta cierta la hipótesis H_0. Análogamente:

Error II:

$$P_1\left(\frac{1}{N}\sum_{i=1}^{N}\log\frac{p(X_i \mid H_1)}{p(X_i \mid H_0)} < \frac{1}{N}\log\frac{P(H_0)}{P(H_1)}\right) \tag{3.33}$$

Pasamos ahora a enunciar, sin demostración, un resultado fundamental de la Teoría de Grandes Desviaciones en el que nos basaremos para el resto del desarrollo.

Teorema de Chernoff extendido [Bahadur 71]:

Sea Y una variable aleatoria real extendida tal que $P(-\infty \le Y < \infty) = 1$. Sea $f(u)$, u variable real, definida por $\exp[-f(u)] = \inf\{e^{-tu}\varphi(t) : t \ge 0\}$, $0 \le f \le \infty$, con $\varphi(t) = E(e^{tY}) = \int_R e^{tY} dF \quad -\infty < t < \infty$, la función generadora de momentos de la función de distribución $F(y)$. Sean $Y_1, Y_2, \ldots$ una secuencia de réplicas independientes de Y, y sean $u_1, u_2, \ldots$ una secuencia de constantes tales que $lim_{N\to\infty} u_N = u, \quad -\infty < u < \infty$ y $P(Y > u) > 0$.

Entonces,

$$lim_{N\to\infty} \frac{1}{N} \log P(Y_1 + \ldots + Y_N \ge N u_N) = -f(u)$$

Particularizando para nuestro problema, si hacemos $Y_i = \log \frac{p(X_i \mid H_1)}{p(X_i \mid H_0)}$ y $u_N = \frac{1}{N} \log \frac{P(H_0)}{P(H_1)}$, entonces $u = 0$ independientemente de los valores de las probabilidades a priori de las hipótesis, y lo anterior se podrá escribir como:

$$lim_{N\to\infty} \frac{1}{N} \log P_0\left(\frac{1}{N} \sum_{i=1}^{N} \log \frac{p(X_i \mid H_1)}{p(X_i \mid H_0)} \ge \frac{1}{N} \log \frac{P(H_0)}{P(H_1)} \right) = -f(0) \qquad (3.34)$$

Si como se ha supuesto, las X_i son variables aleatorias que se distribuyen siguiendo una distribución normal, la expresión anterior nos indica que a medida que aumenta el tamaño N de las muestras, el error tipo I tiende a cero como una exponencial de exponente $-f(0)$. Al mismo valor del exponente puede llegarse para el error tipo II y por lo tanto $-f(0)$ nos da la velocidad exponencial con la que el error promedio de Bayes se va a cero.

Más explícitamente, este exponente puede escribirse como (cambiando inf por $-\sup$):

$$f(0) = \sup_t - \left(\log \int \left(\frac{p(x \mid H_1)}{p(x \mid H_0)} \right)^t p(x \mid H_0) dx \right) \qquad (3.35)$$

A este número se le conoce también como distancia de Chernoff $d_c(p(x \mid H_1), p(x \mid H_0))$ entre las distribuciones $p(x \mid H_1)$ y $p(x \mid H_0)$, y puede considerarse como una medida de "separación" entre ambas.

Es importante también destacar la importancia que tiene trabajar con variables aleatorias extendidas que puedan tomar valores en $-\infty$, ya que de otra manera no sería posible incluir distribuciones de soportes diferentes, lo que supondría una seria limitación.

Para hacernos una idea de las distancias de Chernoff entre las distribuciones de Johnson y la distribución normal, se muestra la gráfica de la figura 3.24.

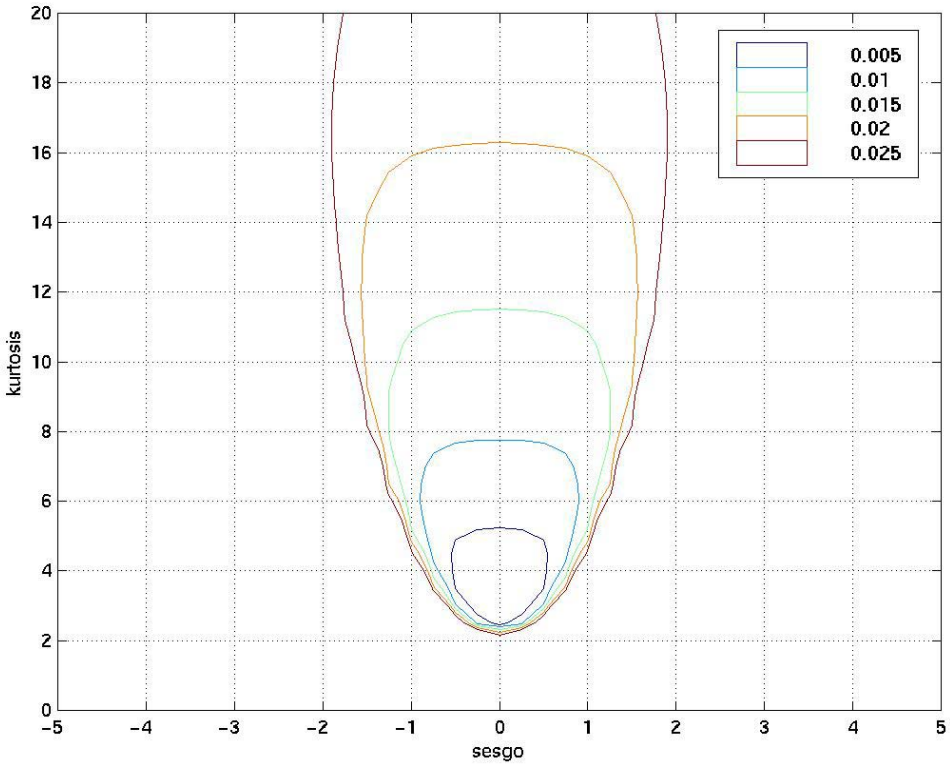

Figura 3.24. Contornos correspondientes a diferentes valores de las distancias de Chernoff en el plano sesgo-kurtosis de distribuciones de Johnson

En esta gráfica aparecen marcadas diferentes líneas de nivel correspondientes a distancias de Chernoff constantes entre la distribución normal (el punto de coordenadas $(0,3)$) y las alternativas de Johnson. En definitiva, para cualquier distribución que esté

sobre una misma curva, el error de Bayes calculado entre esa distribución y la normal, decaerá a cero exponencialmente con el ritmo indicado por la distancia de Chernoff correspondiente cuando $N \to \infty$.

En la discusión que sigue resultará conveniente denominar "experto" a cualquier cociente del tipo $\frac{P(H_1)p(x \mid H_1)}{P(H_0)p(x \mid H_0)}$ y utilizaremos la distancia de Chernoff entre $p(x \mid H_1)$ y $p(x \mid H_0)$, como una medida de lo bueno que es el experto para discriminar entre las dos distribuciones, independientemente de los valores a priori de $P(H_1)$ y $P(H_0)$ y del tamaño de una muestra particular. De gran interés puede resultar también comprobar cuanto de bien puede hacerlo un experto cuando entran en juego datos procedentes de una tercera distribución $p(x \mid H_2)$. Más concretamente, lo que se plantea es qué valor tomará el error tipo II (el error tipo I que se calcula con muestras normales seguirá siendo el mismo) calculado como:

$$P_2\left(\frac{1}{N}\sum_{i=1}^{N}\log\frac{p(X_i \mid H_1)}{p(X_i \mid H_0)} < \frac{1}{N}\log\frac{P(H_0)}{P(H_1)}\right) \tag{3.36}$$

Como un requisito de mínimos para considerar al experto como aceptable, se le deberá exigir que al menos haga el error cero cuando $N \to \infty$. Para encontrar la solución a esta cuestión podemos hacer uso de nuevo de la Teoría de Grandes Desviaciones.

Empezaremos con un teorema [Bahadur 71] que enunciamos sin demostración:

Teorema: Sean Y y $f(u)$ como se definieron en el enunciado del Teorema de Chernoff. Si $E(Y)$ existe y $0 \le E(Y) \le \infty$, entonces, $f(0) = 0$.

Es importante resaltar, que en el caso de cumplirse los supuestos de este teorema, nos encontraríamos con la desagradable sorpresa de que en lugar de tener un decrecimiento exponencial de los errores hacia cero, los errores se incrementarían con el tamaño de la muestra hasta llegar a uno. Veamos que tendría que ocurrir para que se diera este fenómeno.

Definimos primero la distancia de Kullback-Leibler [Kullback 68] entre dos distribuciones $p(x)$ y $q(x)$ como:

$$D_{kl}(q,p) = \int q(x) \log \frac{p(x)}{q(x)} dx \tag{3.37}$$

con $D_{kl}(q,p) = \infty$ si q no es absolutamente continua con respecto a p.

Esta es una medida muy utilizada de la "separación" entre dos distribuciones y puede demostrarse que $0 \le D_{kl}(q,p) \le \infty$, con $D_{kl}(q,p) = 0$ si y sólo si $p(x) = q(x)$.

Particularizando de nuevo para nuestro problema y fijándonos ahora en el error tipo II, tenemos que el teorema anterior puede reformularse como:

Si $0 \le \int \log \frac{p(x \mid H_0)}{p(x \mid H_1)} p(x \mid H_1) dx \le \infty$, entonces, $f(0) = 0$.

Es fácil ver que la expresión de la integral es igual a: $-D_{kl}(p(x \mid H_1), p(x \mid H_0))$, cantidad siempre negativa excepto en el caso carente de interés $p(x \mid H_1) = p(x \mid H_0)$, en que es igual a cero. Por lo tanto, en el cálculo ordinario del error tipo II, no cumpliremos las condiciones bajo las cuales el teorema es válido. Sin embargo, si como ya apuntábamos anteriormente, nos planteamos calcular este error promediando con una tercera distribución $P(x \mid H_2)$, entonces tenemos que:

$$\int \log \frac{p(x \mid H_0)}{p(x \mid H_1)} p(x \mid H_2) dx = \int p(x \mid H_2) \log \frac{p(x \mid H_2)}{p(x \mid H_1)} dx - \int p(x \mid H_2) \log \frac{p(x \mid H_2)}{p(x \mid H_0)} dx =$$
$$= D_{kl}(p(x \mid H_2), p(x \mid H_1)) - D_{kl}(p(x \mid H_2), p(x \mid H_0))$$

Luego, en este caso, la diferencia de distancias puede tomar cualquier valor y por lo tanto podríamos encontrarnos en la anómala situación, ya comentada, de que el error tendiera a uno, a medida que se incrementa el tamaño de la muestra.

La expresión anterior, escrita en términos de las distancias de Kullback-Leibler, proporciona una interpretación más geométrica y sencilla, de tal manera que el experto

para las distribuciones $p(x \mid H_1)$ y $p(x \mid H_0)$, lo hará aceptablemente bien para $p(x \mid H_2)$, si $p(x \mid H_2)$ está más "alejada" de $P(x \mid H_0)$ que de $P(x \mid H_1)$, lo cual se corresponde bastante bien con la idea intuitiva que uno podría hacerse sobre esta cuestión. En definitiva, hemos encontrado la forma de conocer a priori la capacidad de un experto para discriminar con distribuciones diferentes a las óptimas, sin más que evaluar esta diferencia de distancias.

Queda por plantear lo que ocurriría si en lugar de un solo experto tuviéramos varios. Esta situación representa el caso, más interesante para nuestro problema, en el que el cociente $\frac{P(H_1 \mid x)}{P(H_0 \mid x)}$ puede ser escrito en la forma:

$$\frac{P(H_1)P(H_{11})P(x \mid H_{11})}{P(H_0)P(x \mid H_0)} + \frac{P(H_1)P(H_{12})P(x \mid H_{12})}{P(H_0)P(x \mid H_0)} + \cdots + \frac{P(H_1)P(H_{1r})P(x \mid H_{1r})}{P(H_0)P(x \mid H_0)} \tag{3.38}$$

donde cada uno de los sumandos puede ser considerado como un experto, siendo el resultado final la suma de las contribuciones de todos ellos. Puede demostrarse que todo lo dicho anteriormente es aplicable también al conjunto de expertos, sin más que tener en cuenta que ahora el exponente de error vendrá dado por el exponente más pequeño (en valor absoluto) de cada uno de los expertos considerados por separado. La combinación de expertos permite también alcanzar de forma más sencilla, la condición de mínimos a la que hacíamos alusión con anterioridad, ya que basta con que al menos para uno de ellos, se cumpla que el error tienda a cero cuando $N \to \infty$.

Para poner de manifiesto estas cuestiones, examinemos un caso práctico que ayudará a clarificar lo anterior y del que sacaremos importantes conclusiones que se aplicarán en lo que sigue.

Supongamos que $p(x \mid H_0) = N(0,1)$, $p(x \mid H_1) = (1/2)J(0,4) + (1/2)J(0,10)$, donde N y J se refieren a la distribución normal y de Johnson, respectivamente, y $P(H_0) = P(H_1) = 1/2$. De acuerdo con lo expuesto, tendríamos entonces los expertos: $\frac{(1/4)J(0,4)}{(1/2)N(0,1)}$ y $\frac{(1/4)J(0,10)}{(1/2)N(0,1)}$. Se han calculado los errores promedio de Bayes para los dos expertos juntos y con cada uno por separado. Para ello se fue aumentando el tamaño

de la muestra de diez en diez hasta llegar a quinientos, utilizando doscientos cincuenta mil patrones con cada tamaño. Los resultados obtenidos se muestran en la figura 3.25.

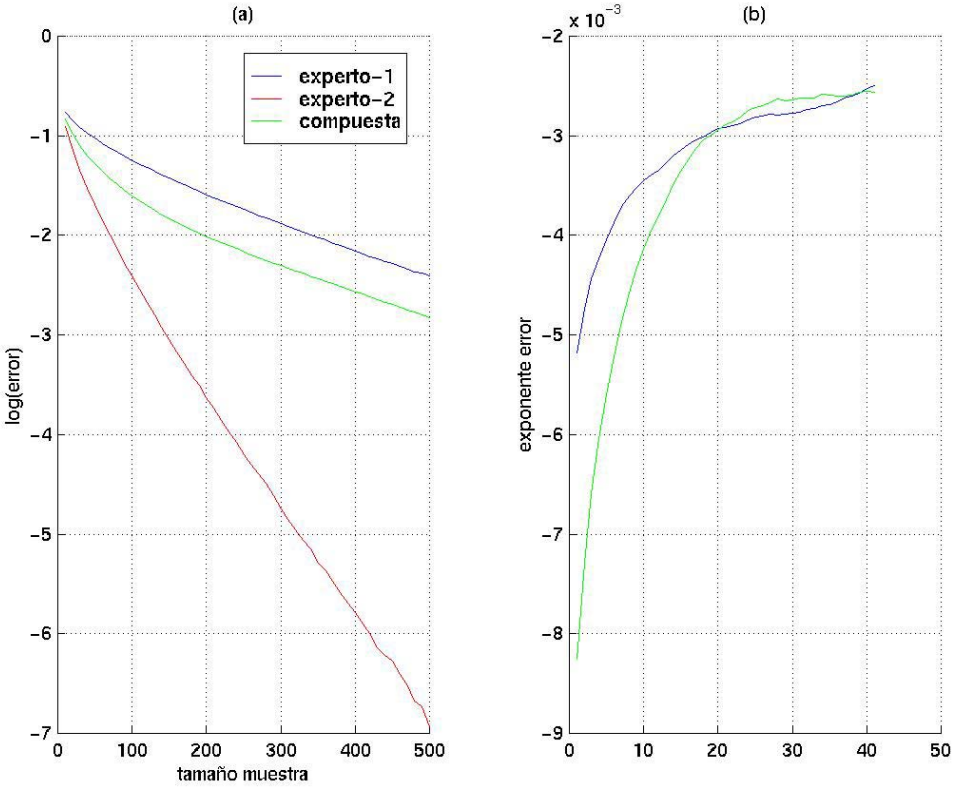

Figura 3.25. (a) Evolución del error promedio, (b) evolución de los exponentes de error

Lo que más llama la atención en la primera de las gráficas, es la apariencia de rectas que tienen las curvas que representan el logaritmo de los errores. Además queda claro que la pendiente de la recta correspondiente a los dos expertos coincide con la pendiente de la recta de uno de ellos, concretamente la de menor pendiente (menor exponente de error).

En la segunda gráfica, que muestra la evolución de las pendientes con el tamaño de las muestras, se aprecia más claramente este hecho. Esto viene a confirmar lo que predecía la Teoría de Grandes Desviaciones respecto al comportamiento exponencial del error cuando $N \to \infty$, si bien, quizás lo más interesante sea comprobar que este comportamiento se manifiesta para valores de N que distan mucho de ∞.

3.5.3 Justificación de los Resultados Obtenidos en los Experimentos Iniciales Usando Teoría de Grandes Desviaciones

Trataremos a continuación de utilizar la teoría anterior para explicar lo observado en los experimentos iniciales cuyos resultados se muestran en la figura 3.23 y en las tablas 3.6 y 3.7.

Habíamos llegado a que para diferentes tamaños de muestra, era posible encontrar tres puntos del plano sesgo-kurtosis de Johnson con los que diseñar un test que diera errores promedio muy similares a los obtenidos con los cinco mil puntos originales. Siguiendo la argumentación del apartado anterior, podemos considerar que lo que tenemos son tres expertos y a modo de ilustración escogeremos los tres correspondientes a $N = 400$. Para cada uno de estos expertos, se ha calculado la diferencia de distancias de Kullback-Leibler $D_{kl}(p(x \mid H_2), p(x \mid H_1)) - D_{kl}(p(x \mid H_2), p(x \mid H_0))$, siendo $p(x \mid H_2)$ una distribución de Johnson arbitraria. Los resultados encontrados se muestran en la figura 3.26.

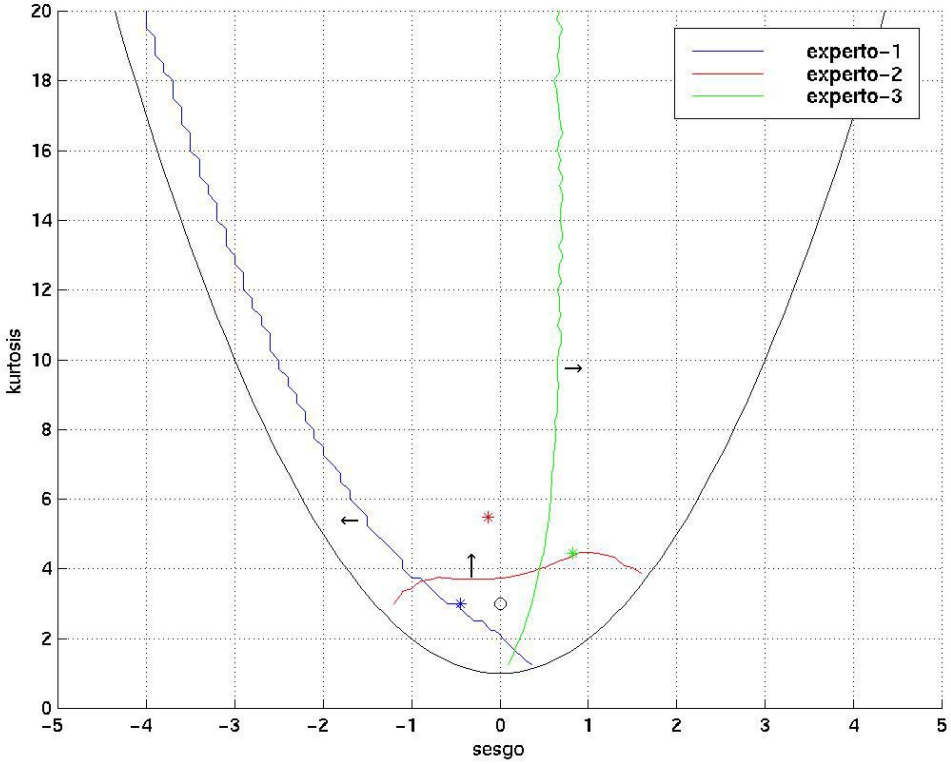

Figura 3.26. Curvas de distancia Kullback-Leibler cero, respecto a la normal, para los expertos (-0.4516,3.0012), (-0.1319,5.4882), (0.8274,4.4497). Las flechas indican en que sentido se comporta bien el experto

Las curvas mostradas determinan las regiones de distribuciones en las que cada experto se comporta "bien" de acuerdo con el criterio, ya explicado, basado en la diferencia de distancias de Kullback-Leibler (las curvas representan las distribuciones $p(x \mid H_2)$ para las que esta distancia se hace cero). Las flechas indican el sentido de este buen comportamiento. Puede apreciarse claramente como los tres expertos parecen suficientes para cubrir satisfactoriamente el plano sesgo-kurtosis de distribuciones de Johnson, exceptuando una pequeña región en torno al punto $(0,3)$ que podemos considerar aceptable, ya que para los tamaños de muestra con los que trabajamos, difícilmente podremos distinguir las distribuciones de esta región de la distribución normal.

De esta forma tan gráfica, podríamos justificar los resultados encontrados, a pesar de que al igual que ocurría en el ejemplo del apartado anterior, los tamaños de muestra que se manejan son bastante pequeños. Todo parece apuntar a que el comportamiento que cabría esperar de los expertos para valores de $N \to \infty$, es aplicable en buena medida para valores de N sorprendentemente pequeños.

3.5.4 Elección de Expertos Informativos

En el apartado anterior, se demostró que tres expertos bastaban para abarcar todo el plano de distribuciones de Johnson. Estos expertos se obtuvieron como resultado de un proceso de minimización, en el que no se imponía ningún criterio adicional que restringiera su selección.

Trataremos, ahora, de escoger a los expertos de forma que resulten, además de eficientes, informativos del tipo de desviación de la normalidad con la que nos enfrentamos.

El procedimiento seguido es similar al seguido en los experimentos iniciales, de hacer subconjuntos, con la diferencia de que se distinguieron dos zonas dadas por: $\sqrt{\beta_1} > 0, \quad 1 < \beta_2 < 3$; $\sqrt{\beta_1} > 0, \quad 3 < \beta_2 < 10$. En cada una se eligió una única distribución (subconjunto con un solo elemento) que hacía el error promedio lo más parecido al error medido considerando todos los puntos de la región. Para obtener los

otros dos puntos correspondientes a las regiones simétricas con $\sqrt{\beta_1} < 0$, simplemente se cambió el signo de la componente de sesgo. Hay que decir que todo este cómputo se realizó para $N = 400$. Así se obtuvieron los cuatro puntos: (0.2717,2.6298), (0.4697,4.4994), (-0.2717,2.6298), (-0.4697,4.4994), lo que nos permite diseñar un test combinando los correspondientes cuatro expertos.

Lo primero que había que comprobar, al igual que se hizo en el caso de los tres expertos, era si esta combinación bastaba para dar cuenta del grueso del conjunto de distribuciones de Johnson, o por el contrario, quedaban zonas importantes por cubrir. La figura 2.27 aclara esta cuestión.

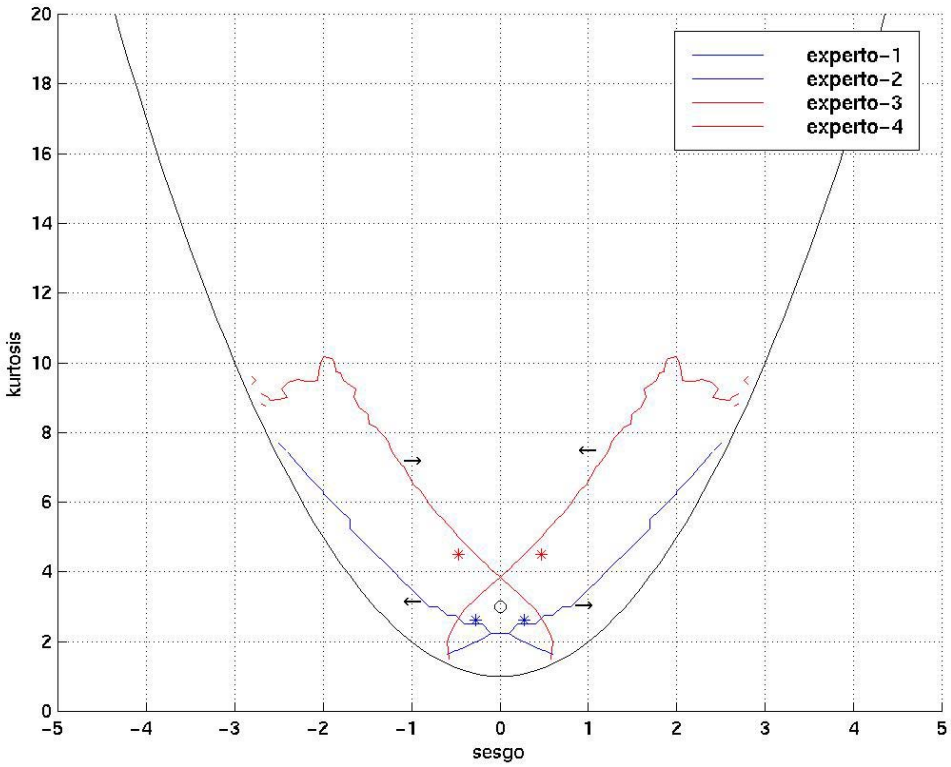

Figura 3.27. Curvas de distancia Kullback-Leibler cero, con respecto a la normal, para los expertos (0.2717,2.6298), (0.4697,4.4994), (-0.2717,2.6298), (-0.4697,4.4994). Las flechas indican en que sentido se comporta bien el experto

De nuevo, parece que los cuatro expertos son suficientes para garantizar que el error se irá a cero para cualquier distribución exceptuando una pequeña región en torno a la distribución normal.

Pasemos ahora a tratar de verificar si nuestra elección resulta realmente "informativa" y en qué sentido lo es.

Para la discusión que sigue, resultará conveniente escribir la probabilidad a posteriori de ser no normal en la forma:

$$P(H_1 \mid x) = \frac{P(H_1)p(x \mid H_1)}{p(x)} = \frac{P(H_1)P(H_{11})p(x \mid H_{11})}{p(x)} + \cdots + \frac{P(H_1)P(H_{1r})p(x \mid H_{1r})}{p(x)} \tag{3.39}$$

donde podemos considerar que tenemos diferentes componentes asociadas a cada uno de los expertos. Cada una de estas componentes supone una contribución a la probabilidad a posteriori y por lo tanto estará acotada entre 0 y 1. Como los cuatro expertos anteriores han sido escogidos por su buen comportamiento en regiones comprendiendo un rango de valores de sesgo y kurtosis determinados, concretamente,

$(0.2717,2.6298)$: $\sqrt{\beta_1} > 0, \quad 1 < \beta_2 < 3$,

$(0.4697,4.4994)$: $\sqrt{\beta_1} > 0, \quad 3 < \beta_2 < 10$,

$(-0.2717,2.6298)$: $\sqrt{\beta_1} < 0, \quad 1 < \beta_2 < 3$,

$(-0.4697,4.4994)$: $\sqrt{\beta_1} < 0, \quad 3 < \beta_2 < 10$,

los valores que tomen las diferentes componentes de la probabilidad a posteriori, pueden ser significativas del tipo de desviación de la normalidad presente. Así, si por ejemplo, la componente asociada al experto $(0.2717,2.6298)$ diera un valor muy superior al resto, cabría esperar que la muestra provenga de una distribución con sesgo positivo y kurtosis menor que la de la normal. Esta información podría servir de ayuda de cara a intentar normalizar la muestra, si se creyera conveniente. En las tablas 3.8 y 3.9, figuran los valores de media y varianza de las diferentes componentes, calculados utilizando mil muestras de tamaños 25 y100, y diferentes distribuciones de referencia.

	$(0.2717,2.6298)$	$(0.4697,4.4994)$	$(-0.2717,2.6298)$	$(-0.4697,4.4994)$
$(0,3)$	0.1386/0.0096	0.1025/0.0064	0.1407/0.0104	0.1065/0.0072
$(0,2)$	0.2031/0.0078	0.0557/0.0013	0.2077/0.0076	0.0581/0.0017
$(0,10)$	0.0832/0.0092	0.2070/0.0441	0.0766/0.0075	0.2013/0.0427
$(1,3)$	0.3077/0.0098	0.2959/0.0369	0.0242/0.0015	0.0179/6.3e-5
$(-1,3)$	0.0258/0.0016	0.0181/6.4e-5	0.3120/0.0093	0.2856/0.0355
$(1,10)$	0.1130/0.0106	0.2968/0.0612	0.0501/0.0054	0.1168/0.0162
$(-0.5,2)$	0.0998/0.0063	0.0226/1.2e-4	0.3382/0.0040	0.1204/0.0102

Tabla 3.8. Medias y varianzas de las componentes de la probabilidad a posteriori para N=25

	$(0.2717,2.6298)$	$(0.4697,4.4994)$	$(-0.2717,2.6298)$	$(-0.4697,4.4994)$
$(0,3)$	0.1001/0.0312	0.0760/0.0142	0.1097/0.0340	0.0837/0.0164
$(0,2)$	0.4070/0.0748	0.0016/7.7e-6	0.4210/0.0742	0.0015/6.1e-6
$(0,10)$	0.0032/5.0e-4	0.4366/0.1375	0.0061/0.0015	0.3819/0.1276
$(1,3)$	0.6570/0.0980	0.3304/0.1004	9.0e-6/8.4e-9	3.7e-6/3.8e-11
$(-1,3)$	1.6e-5/4.3e-8	3.8e-6/6.2e-11	0.6426/0.0990	0.3451/0.1015
$(1,10)$	0.0162/0.0049	0.7506/0.0949	0.0014/1.6e-4	0.0932/0.0379
$(-0.5,2)$	0.0097/0.0014	9.5e-6/9.7e-10	0.9585/0.0028	0.0079/9.7e-4

Tabla 3.9. Medias y varianzas de las componentes de la probabilidad a posteriori para N=100

Como se puede comprobar, en ambos casos, y especialmente para $N = 100$, los valores-Promedio encontrados son indicativos, para unas alternativas más claramente que para otras, del tipo de desviación de la normalidad. Por ejemplo, cuando las muestras vienen de la distribución J(-0.5,2), se destaca enormemente el valor de la componente asociada al experto (-0.2717,2.6298), esto es, sesgo negativo y kurtosis por debajo de tres, justamente la misma región a la que pertenece la alternativa considerada.

Queda por comprobar si, además de informativos, los expertos seleccionados son igualmente eficientes. Para ello se realizarán una serie de experimentos similares a los que se hicieron con las redes neuronales.

En primer lugar, en la tabla 3.10 se muestran los errores promedio calculados con los mismos patrones que los de la tabla 3.5.

N	*Errores promedio*
10	0.4402
25	0.3917
50	0.3238
100	0.2156
200	0.1175
400	0.0500

Tabla 3.10

Se observa como a partir de $N = 100$ empiezan a parecerse ambos resultados, siendo casi iguales para $N = 400$.

Se calcularon también las potencias de rechazo frente los conjuntos de distribuciones 1,2 y 3, empleados con las redes. En las figuras 3.28, 3.29 y en la tabla 3.11 se muestran los resultados encontrados.

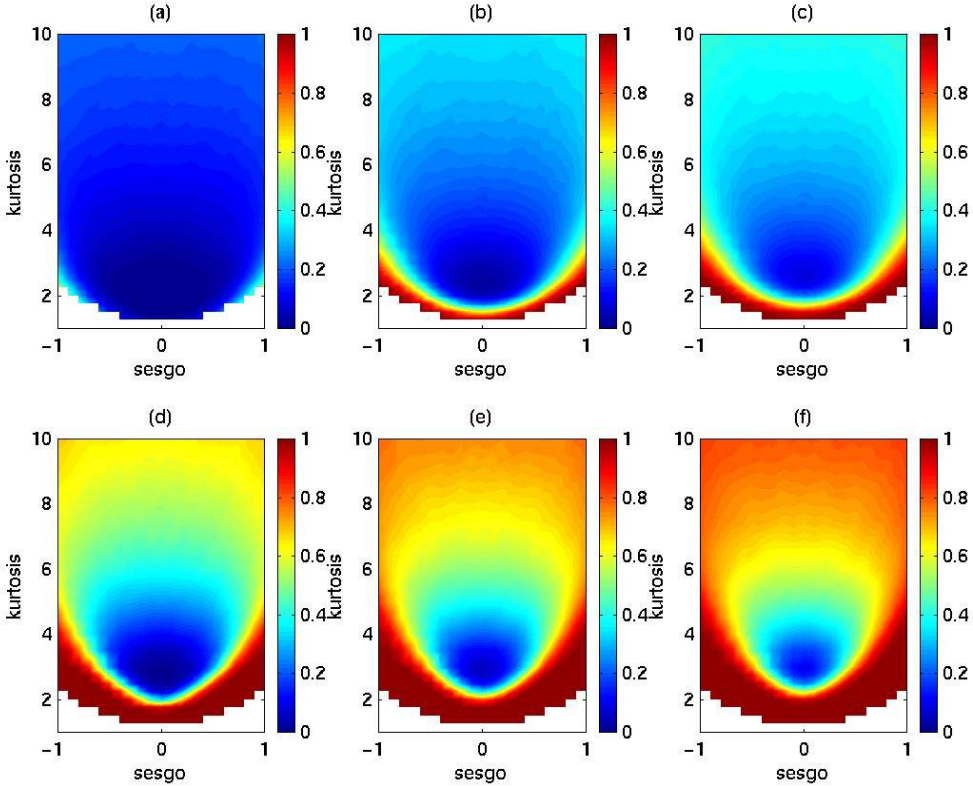

Figura 3.28. Potencias de rechazo de los expertos a las distribuciones de Johnson para (a) N=25, nivel 0.01, (b) N=25, nivel=0.05, (c) N=25, nivel=0.1, (d) N=100, nivel 0.01, (e) N=100, nivel 0.05, (f) N=100, nivel=0.1

Excepto en el caso N=25 y nivel 0.01, en el resto, los resultados encontrados son comparables a los ya vistos para otros tests y las redes neuronales, observándose mejoría en la zona de kurtosis inferior a tres.

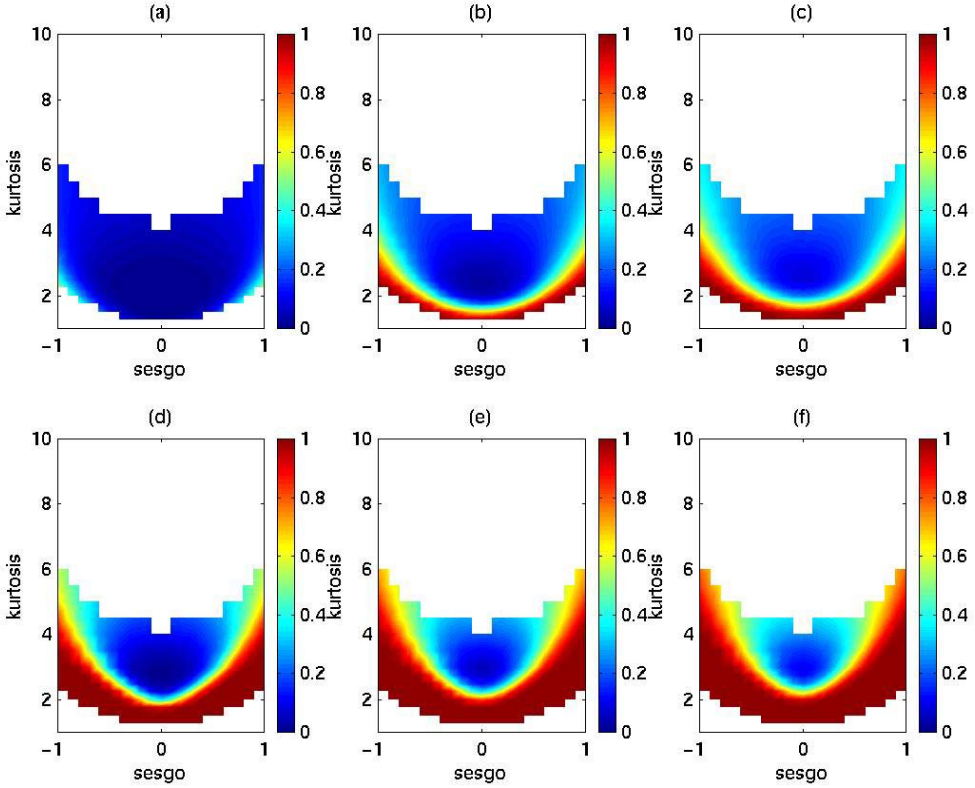

Figura 2.29. Potencias de rechazo de los expertos a las distribuciones de Pearson para (a) N=25, nivel 0.01, (b) N=25, nivel=0.05, (c) N=25, nivel=0.1, (d) N=100, nivel 0.01, (e) N=100, nivel 0.05, (f) N=100, nivel=0.1

Al igual que ocurría con las redes neuronales, los resultados encontrados son similares a los de las distribuciones de Johnson.

Finalmente, utilizaremos el conjunto 3 de distribuciones para evaluar las prestaciones de nuestro test basado en la combinación de expertos. En la tabla 3.11 se muestran los resultados encontrados.

	$Expertos(25,0.01)$	$Expertos(100,0.1)$
Uniforme	0.0000	0.9985
χ^2_2	0.5681	1.0000
Weibull-2	0.0475	0.9034
Mezcla-Normales	0.0000	0.9971

Tabla 3.11

Si bien, para N=25 y nivel 0.01, los resultados no siempre son satisfactorios (como ya habíamos constatado en las figuras 2.28 y 2.29), para N=100 y nivel 0.1, los resultados son excelentes, mejores que los de cualquiera de los procedimientos ya probados.

3.5.5 Estimación de las Probabilidades a Posteriori dadas por la Combinación de Expertos

Una de las características que destacamos de las redes neuronales, era su capacidad para estimar, con gran precisión, las probabilidades a posteriori asociadas a las distribuciones normales y no normales. Esta nueva aproximación al problema, también ofrece esta posibilidad, incluso de forma más directa que con la anterior.

Para poner este hecho de manifiesto y de paso comparar con los valores-P que se obtienen de tests como el de Shapiro-Wilk, hemos realizado un sencillo experimento en el que se promedian ambas magnitudes sobre un conjunto de cinco mil muestras normales cuyo tamaño vamos variando de veinticinco a quinientos. En la figura 3.30, se muestra una gráfica representativa de la simulación realizada.

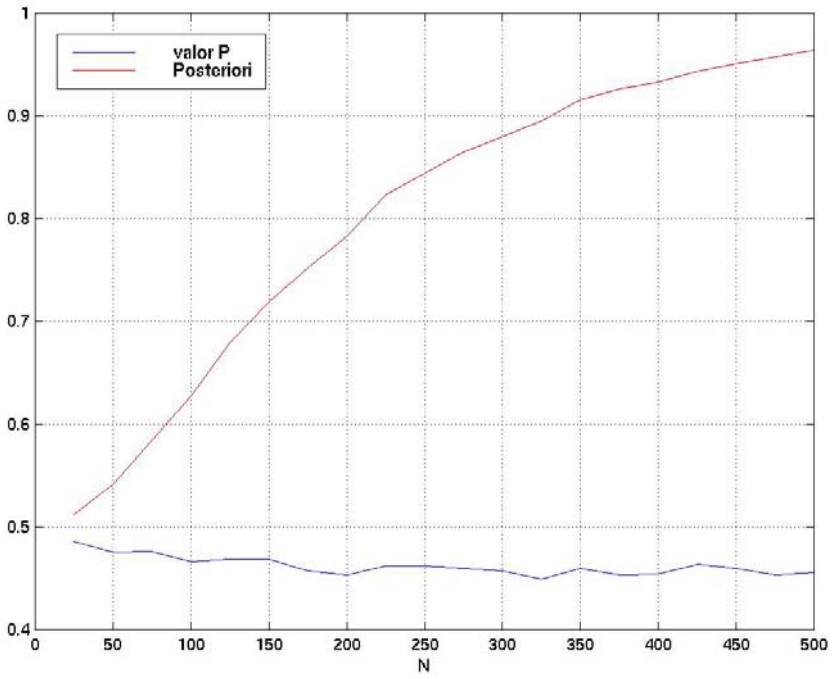

Figura 3.30. Variación de la probabilidad a posteriori y del valor-P del test de Shapiro-Wilk con el tamaño de muestra

Los resultados encontrados son realmente significativos, porque si bien para $N = 25$, ambas curvas parten de un valor similar, el valor-P se mantiene más o menos estable, mientras que la probabilidad a posteriori, estimada para la combinación de expertos, aumenta hasta valores cercanos a uno, comportamiento lógico y deseable, ya que la incertidumbre debe disminuir a medida que aumenta el tamaño de la muestra. Una vez más, queda claro que los valores-P no reflejan de forma fiable el soporte que los datos dan a las hipótesis.

A modo de conclusión, decir que los procedimientos desarrollados en este capítulo son aplicables a otras distribuciones diferentes de la normal, que también puedan resultar interesantes por sus aplicaciones, como por ejemplo la distribución exponencial.

CAPÍTULO 4. DISEÑO DE UN CLASIFICADOR

El procedimiento usual para diseñar un clasificador suele comenzar con una inspección visual de los datos, tratando de intuir la dificultad del problema y qué modelo de clasificador podría resultar el más adecuado. Este procedimiento, aparte de resultar un tanto "subjetivo", tiene el inconveniente añadido de la dificultad para la visualización en espacios de tres o más dimensiones, que por otro lado suele ser lo habitual. Sería interesante, por lo tanto, basar el proceso de diseño en medidas más cuantificables y objetivas. Con este fin, se plantea la construcción de un Sistema Basado en el Conocimiento que incorpore el conocimiento necesario para llevar a cabo dicho proceso. Aplicando la metodología CommonKADS, se planteará el Modelo de Conocimiento del problema, y se implementará siguiendo los pasos indicados en el capítulo 2.

El diseño de un clasificador es una tarea muy compleja. En este dominio es infrecuente encontrar problemas en los que se conozcan reglas específicas que puedan ser usadas en el diseño. Como ejemplo, podríamos citar el caso del diseño de

compiladores que pueden clasificar programas escritos, en un determinado lenguaje, como correctos o incorrectos, sin necesidad de observar ningún programa previo. Esto es posible, gracias a que los patrones de información están expresados a través de una gramática formal que puede ser claramente identificada por el clasificador. Tendríamos entonces lo que se denominan reconocedores de patrones sintácticos, si bien, son de uso muy restringido.

En la gran mayoría de aplicaciones, no se hacen suposiciones estructurales, y toda la estructura del clasificador se aprende a partir de los datos disponibles. Es lo que se conoce como reconocimiento de patrones estadístico. Es fundamental, por lo tanto, obtener la máxima información de estos datos, a menudo escasos. En este sentido, cobran especial importancia, las técnicas de análisis orientadas a averiguar, en la medida de lo posible, las distribuciones de probabilidad de los patrones. Por este motivo, se han integrado en el Sistema Basado en el Conocimiento, las técnicas desarrolladas en el capítulo anterior.

Finalmente, el sistema desarrollado ha sido probado con diferentes tipos de datos, procedentes, en su mayoría, del entorno de la medicina. Algunos de ellos se encuentran en bases de datos de libre acceso, como pueden ser los conocidos datos Iris de Fisher, y otros se han obtenido a partir de la colaboración del grupo de Computadoras y Control de la Universidad de La Laguna con entidades locales. Así, se aplicaron las técnicas desarrolladas a datos de pacientes con demencias como el Alzheimer, recogidos en el Departamento de Neurofisiología del Hospital Nuestra Señora de la Candelaria. Por otro lado, se dispuso de datos de niños con Dislexia, fruto de la colaboración con el Departamento de Psicología Evolutiva y de la Educación de la Universidad de La Laguna.

4.1 El Diseño de un Clasificador: Una Tarea Compleja

La complejidad de la tarea de diseñar un clasificador para un problema particular viene dada por diversos factores, que combinados, generan una gran incertidumbre en el diseñador, debido al amplio abanico de posibilidades que se originan. Entre estos factores cabe destacar los siguientes:

- Preprocesamiento de los datos

- Elección del modelo de clasificador. El problema de la generalización.
- Determinación de los parámetros de un clasificador.
- El problema de la dimensionalidad.
- Evaluación de las prestaciones de un clasificador.

Hay que decir que estos factores están totalmente interrelacionados, lo cual complica aún más el problema. Dada la importancia que tienen en el desarrollo posterior, nos referiremos a cada uno de ellos en las secciones que siguen.

Comentar también que no se pretende dar una visión exhaustiva de toda la problemática que rodea el diseño de un clasificador, aunque si lo suficientemente amplia para hacerse una idea de su complejidad. Además, se ha insistido especialmente en aquellas cuestiones que se han tratado más a fondo en este trabajo.

4.1.1 Preprocesamiento de los Datos

En la práctica, es frecuente, aplicar a los datos algún tipo de transformación, antes de ser utilizados en el proceso de diseño, propiamente dicho.

Una de las formas más comunes de preprocesamiento consiste en un simple reescalado lineal de las variables de entrada. Esto es importante, porque, a menudo, diferentes variables o características difieren notablemente a causa de las unidades en que han sido medidas, y no por su importancia relativa discriminatoria en la clasificación. El escalado más simple consiste en:

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i , \qquad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(X_i - \overline{X}\right)^2 ,$$

$$\widetilde{X}_i = \frac{X_i - \overline{X}}{\sigma} \tag{4.1}$$

De gran importancia son, sin duda, las técnicas de preprocesamiento destinadas a reducir la dimensionalidad de los datos de entrada. De todas formas, se pospone la discusión sobre estas técnicas a la sección 4.1.4 dedicada exclusivamente a esta cuestión.

El problema de la ausencia de algunos datos (*missing values* [Little 87]), ha sido ignorado durante mucho tiempo en la literatura acerca del reconocimiento de patrones.

En dominios como el diagnóstico médico, es frecuente que falten valores de algunas de las características, por ejemplo, si un médico decide no realizar un test cuyo resultado parece seguro o no es relevante para el diagnóstico. También podría ocurrir que se tratara de una característica muy difícil de medir. En otros dominios, en los que los datos se obtienen por algún procedimiento automático, es más raro que se produzca este hecho.

Se han planteado diferentes soluciones a este problema, algunas tan simples como reemplazar los valores que faltan por valores "típicos" como el promedio sobre valores observados, y otras más sofisticadas [Ghahramani 94].

4.1.2 Elección del Modelo de Clasificador. El Problema de la Generalización

La primera decisión a tomar en el diseño de un clasificador será con toda seguridad la elección del modelo con el que se va a trabajar. Como ya se explicó en la parte de fundamentos, existe una amplia variedad de modelos de clasificadores, entre los cuales podemos citar: modelos paramétricos lineales y cuadráticos, más flexibles como redes neuronales, no paramétricos como los clasificadores de vecinos próximos, árboles de decisión, ...,etc

Una de las cuestiones claves relacionadas con la elección del modelo es, sin duda, su complejidad. El hecho de disponer de un conjunto finito de datos de entrenamiento, condiciona en gran manera el proceso de diseño, ya que, no se trata de conseguir que el sistema memorice estos datos, sino que sea capaz de hacer buenas predicciones con otros datos no presentes en el entrenamiento, esto es, que generalice adecuadamente.

Para expresar, más formalmente, cómo un número de datos finito N condiciona la complejidad que debe tener un modelo, elegiremos el criterio del error cuadrático, para medir la bondad de un clasificador, esto es,

$$E = \frac{1}{2}\sum_{j=1}^{N}\sum_{i=1}^{L}\left\{y_i\left(X_{i,j};\theta\right) - t_i^j\right\}^2 \tag{4.2}$$

donde, t_i^j es la etiqueta de la clase correspondiente a cada dato $X_{i,j}$ (codificada con algún esquema adecuado como el 1 de L) y $y_i(X_{i,j};\theta)$ representa la salida de un modelo de clasificador de parámetros θ.

En el caso ideal $N=\infty$, esta función de error puede expresarse como [Bishop 95]:

$$E=\frac{1}{2}\sum_{i=1}^{L}\int\{y_i(x;\theta)-\langle t_i\mid x\rangle\}^2\,p(x)dx+\frac{1}{2}\sum_{i=1}^{L}\int\{\langle t_i^{\,2}\mid x\rangle-\langle t_i\mid x\rangle^2\}p(x)dx \qquad (4.3)$$

donde, $p(x)$ es la densidad incondicional de los datos de entrada; y $\langle t_i\mid x\rangle$ denota el promedio condicionado $\langle t_i\mid x\rangle=\int t_i\,p(t_i\mid x)dt_i$, de hecho, la probabilidad a posteriori de las clases $P(C_i\mid x)$.

De la expresión, se deduce que el clasificador óptimo, en este sentido, será aquel que haga $y_i(x;\theta)=\langle t_i\mid x\rangle=P(C_i\mid x)$. En general, para conseguirlo, bastará con considerar un modelo lo suficientemente flexible que sea capaz de generar cualquier superficie discriminatoria, por ejemplo, una red neuronal con dos capas ocultas de neuronas. A pesar de todo, el error, en general, no será cero ya que el segundo término de la expresión no depende de $y_i(x;\theta)$ y puede tomarse como la dificultad intrínseca del problema.

Desgraciadamente, en la práctica, las cosas resultan mucho más complicadas, ya que, N es finito y, casi siempre, reducido. Para estudiar lo que esto supone en relación a la complejidad, partimos de la expresión

$$\{y_i(x;\theta)-\langle t_i\mid x\rangle\}^2 \qquad (4.4)$$

que como acabamos de ver, da cuenta de lo bueno que es el modelo seleccionado. Lo que ocurre es que, ahora, esta medida va a depender del conjunto particular de datos de entrenamiento que se utilice. Para eliminar esta dependencia, promediaremos sobre todos los conjuntos de entrenamiento D de tamaño N_i, procedentes de la misma distribución $p(x,t_i)$, para calcular

$$\overline{E}_D\left[\{y_i(x;\theta)-\langle t_i\mid x\rangle\}^2\right] \qquad (4.5)$$

donde $\overline{E}_{D[\ ]}$ denota el valor promedio. Parece claro entonces, que el objetivo será elegir un modelo que haga cero esta diferencia promedio. En general, existen dos motivos por los que esta diferencia va a ser diferente de cero. Para ahondar un poco más en esta cuestión, va a resultar conveniente expandir esta expresión de la siguiente manera [Bishop 95]:

$$\overline{E}_D\left[\left\{y_i(x;\theta)-\langle t_i \mid x\rangle\right\}^2\right]=\left\{\overline{E}_D\left[y_i(x;\theta)\right]-\langle t_i \mid x\rangle\right\}^2+\overline{E}_D\left[\left\{y_i(x;\theta)-\overline{E}_D\left[y_i(x;\theta)\right]\right\}^2\right] \tag{4.6}$$

con lo que tenemos dos términos, al primero de los cuales se le denomina sesgo o *bias* y al segundo varianza [Geman 92]. Así, el término de bias refleja, hasta que punto, el promedio de las salidas del clasificador difiere de la función deseada $\langle t_i \mid x\rangle$, mientras que el término de varianza da cuenta de la sensibilidad de $y_i(x;\theta)$ a un conjunto de datos particular.

Como estas expresiones están en función del vector de entrada x, podemos también integrar sobre todos los valores de x para obtener:

$$(bias)^2=\frac{1}{2}\int\left\{\overline{E}_D\left[y_i(x;\theta)\right]-\langle t_i \mid x\rangle\right\}^2 p(x)dx \tag{4.7}$$

$$\text{var}\,ianza=\frac{1}{2}\int\overline{E}_D\left[\left\{y_i(x;\theta)-\overline{E}_D\left[y_i(x;\theta)\right]\right\}^2\right]p(x)dx \tag{4.8}$$

De este planteamiento, se deduce que la clave, en lo que se refiere a la complejidad del modelo, consiste en buscar el equilibrio óptimo entre bias y varianza. Si el modelo elegido es demasiado flexible, se "pegará" mucho a los datos particulares del entrenamiento, y la varianza tenderá a ser alta. Por otro lado, si es muy poco flexible, el sesgo puede hacerse igualmente grande. Esta problemática lleva a que modelos simples y restringidos como los lineales cobren un protagonismo importante, ya que, sobre todo, en aplicaciones en las que se dispone de, relativamente, pocos datos, pueden dar un mejor rendimiento que clasificadores no lineales aún cuando los primeros sean un caso particular de estos.

Esta circunstancia queda claramente reflejada en el conocido fenómeno de Hugues [Hugues 68], que se ilustra, a través de un ejemplo, en la figura 4.1., en la que se muestra como el error promedio de un clasificador cuadrático, entrenado con un número de datos fijo, crece a medida que aumenta la dimensionalidad.

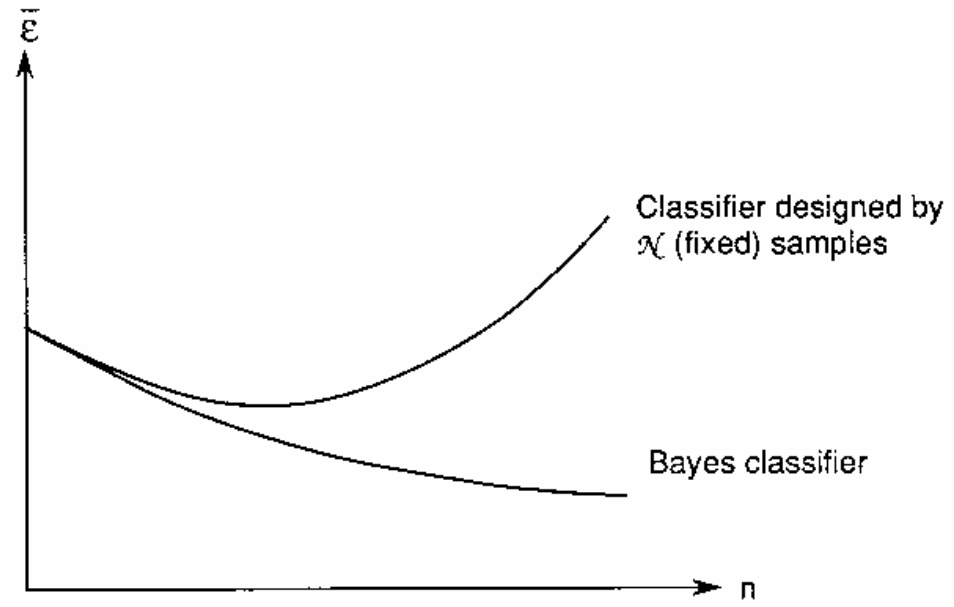

Figura 4.1. Error promedio de un clasificador cuadrático, entrenado con un número fijo de datos (figura extraida de [Fukunaga 90])

Existen diversas técnicas para abordar el problema de la complejidad. Los dos enfoques principales se basan en:

- Seleccionar iterativamente un modelo, tomando como referencia alguna medida indicativa de la bondad del mismo, de tal manera que se aumente o disminuya su complejidad de acuerdo con los resultados encontrados. Por ejemplo, variar el número de pesos en una red neuronal.

- Utilizar un criterio de selección que incluya un término de penalización de la complejidad del modelo. Como ejemplos de criterios, podemos citar el criterio AIC

de Akaike [Akaike 73] [Akaike 74], el enfoque de Vapnik [Vapnik 82] o métodos basados en validación cruzada.

4.1.3 Determinación de los Parámetros de un Clasificador

En la sección anterior, consideramos que teníamos una función discriminante $y_k(x;\theta)$, dependiente de un conjunto de parámetros θ. Estos parámetros pueden ser medias y matrices de covarianza, en discriminantes lineales y cuadráticos, pesos de una red neuronal,... En cualquier caso debemos encontrar un procedimiento que nos permita calcular estos parámetros a partir del conjunto de datos de entrenamiento. Existen diversos métodos para dar valores a θ. Distinguiremos entre aquellos que se utilizan en el contexto de la teoría de la decisión estadística y aquellos que se emplean con sistemas como redes neuronales.

4.1.3.1 Determinación de Parámetros en Clasificadores Estadísticos

En el enfoque puramente estadístico del problema, habitualmente $\theta = (\mu_i, \Sigma_i)$, donde, μ_i y Σ_i representan los vectores de medias y las matrices de covarianza de las clases, respectivamente.

Habitualmente, estas medias y covarianzas son estimadas e insertadas en la expresión de las densidades de probabilidad de las clases, para después calcular las aproximaciones a las probabilidades a posteriori. Al clasificador así diseñado, se le llama un clasificador *plug-in* (sería lo mismo, si en lugar de las densidades se utilizan directamente las probabilidades a posteriori, como en el modelo logístico múltiple). De esta manera, las probabilidades a posteriori se calcularían como:

$$\hat{P}(C_i \mid x) = \frac{P(C_i)p(x;\hat{\theta} \mid C_i)}{\sum_{k=1}^{L} P(C_k)p(x;\hat{\theta} \mid C_k)} \tag{4.9}$$

Queda por decidir que estimador utilizar para determinar los parámetros. El estimador de máxima verosimilitud (*maximum likelihood*), ha sido, desde siempre, la opción

práctica más popular. Por ejemplo, si se asume que las densidades de las clases siguen una distribución normal, esto es,

$$p(x \mid C_i) = (2\pi)^{-n/2} |\Sigma_i|^{-1/2} \exp\left[-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right] \tag{4.10}$$

la verosimilitud en la clase C_i, para un conjunto de entrenamiento de la forma $D = \{X_{i,1}, \ldots, X_{i,N_i}, i = 1, \ldots, L\}$, vendrá dada por:

$$\prod_{j=1}^{N_i} |\Sigma_i|^{-1/2} \exp\left[-\frac{1}{2} (X_{i,j} - \mu_i)^T \Sigma_i^{-1} (X_{i,j} - \mu_i) \right] \tag{4.11}$$

expresión que se maximiza cuando $\hat{\mu}_i = \overline{X}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} X_{i,j}$ y $\hat{\Sigma}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} (X_{i,j} - \hat{\mu}_i)(X_{i,j} - \hat{\mu}_i)^T$, siendo estos, por consiguiente, los respectivos estimadores de máxima verosimilitud.

En el caso particular en el que las matrices de covarianza de todas las clases son iguales, el estimador de máxima verosimilitud vendrá dado por:

$$\hat{\Sigma} = \sum_{i=1}^{L} \frac{N_i}{N} \hat{\Sigma}_i \tag{4.12}$$

donde, $N = \sum_{i=1}^{L} N_i$ y a $\hat{\Sigma}$ se le conoce como matriz de covarianza común.

A pesar de que $\hat{\mu}_i$, $\hat{\Sigma}_i$ y $\hat{\Sigma}$, resultan ser, en general, buenos estimadores de μ_i, Σ_i y Σ, calculados en el límite de infinitos datos, ello no nos garantiza que las densidades estimadas sean buenos estimadores de las verdaderas densidades de probabilidad, que es lo que realmente interesa. Es por esto, que existen otros estimadores que, en ocasiones, pueden resultar más apropiados que los de máxima verosimilitud.

Entre estos estimadores alternativos, pensados para ser utilizados en clasificadores "plug-in", podemos citar los siguientes:

- Modificar los estimadores para hacerlos insesgados o menos sesgados de lo que son originalmente. El caso más sencillo es el del estimador de la matriz de covarianza $\hat{\Sigma}_i$, que, como es bien sabido, es sesgado y por este motivo se suele tomar como denominador $N_i - 1$ en lugar de N_i, quedando:

$$\widetilde{\Sigma}_i = \frac{1}{N_i - 1}\sum_{j=1}^{N_i}\left(X_{i,j} - \hat{\mu}_i\right)\left(X_{i,j} - \hat{\mu}_i\right)^T \tag{4.13}$$

Análogamente para la matriz de covarianza común, tenemos:

$$\widetilde{\Sigma} = \sum_{i=1}^{L}\frac{N_i}{N - L}\hat{\Sigma}_i \tag{4.14}$$

- Otra posibilidad es la utilización de estimadores robustos de medias y covarianzas. Estos estimadores surgen, motivados por dos cuestiones principales. Por un lado, el hecho de que aunque la distribución normal es una abstracción conveniente, estudios más precisos, muestran que las distribuciones reales, en general, no son normales, sino que tienen colas más gruesas. Por otro lado, está el tema de los *outliers*, esto es, datos que no pertenecen a la clase considerada, por ejemplo, porque hayan sido incorrectamente etiquetados. En lo que respecta al asunto de la normalidad, suponer que los datos siguen una distribución t de Student multivariable, parece una hipótesis más fiable. La densidad de una distribución t multivariable con tres o más grados de libertad, puede escribirse de la siguiente manera [Mardia 79]:

$$\frac{\Gamma\left(\frac{1}{2}(\nu + n)\right)}{(\nu\pi)^{n/2}\Gamma\left(\frac{1}{2}\nu\right)}|\Sigma|^{-\frac{1}{2}}\left[1 + \frac{1}{\nu}(x - \mu)^T\Sigma^{-1}(x - \mu)\right]^{-\frac{(\nu + n)}{2}} \tag{4.15}$$

la cual tiene contornos elípticos con forma determinada por Σ, pero que se dispersan más lentamente que en una distribución normal. Si se calculan los estimadores de máxima verosimilitud para esta función de densidad, se obtiene [Huber 81]:

$$\hat{\mu}_i = \sum_{j=1}^{N_i} w_{i,j}\left(\hat{\mu}_i, \hat{\Sigma}_i\right) X_{i,j} \Big/ \sum_{j=1}^{N_i} w_{i,j}\left(\hat{\mu}_i, \hat{\Sigma}_i\right) \tag{4.16}$$

$$\hat{\Sigma}_i = \frac{1}{N_i}\frac{\nu_i + n}{\nu_i} \sum_{j=1}^{N_i} w_{i,j}\left(\hat{\mu}_i, \hat{\Sigma}_i\right)\left(X_{i,j} - \hat{\mu}_i\right)\left(X_{i,j} - \hat{\mu}_i\right)^T \tag{4.17}$$

donde, ν_i representa los grados de libertad y $w_{i,j} = \dfrac{1}{\left(1 + Q_{i,j}/\nu_i\right)}$, con $Q_{i,j} = \left(X_{i,j} - \mu_i\right)^T \Sigma_i^{-1}\left(X_{i,j} - \mu_i\right)$.

Tenemos entonces que estos estimadores son versiones ponderadas de los estimadores de máxima verosimilitud habituales. Por otro lado, si se suponemos normalidad, pero pensamos que puede haber *outliers*, tendremos también que usar estimadores robustos. Algunos métodos recientes se basan en ajustar la escala de la covarianza, eliminando así puntos no deseados. En concreto, existe un método que encuentra el elipsoide de volumen mínimo conteniendo $\left[\left(N_i + n + 1\right)/2\right]$ puntos. Este elipsoide se usa para definir una distancia de Mahalanobis, y todos los puntos que están a una distancia del centro del elipsoide de hasta un 97.5%, se retienen [Rousseeuw 90].

- Hemos visto los estimadores de máxima verosimilitud de las matrices de covarianza, en el caso general, $\hat{\Sigma}_i$, y en el caso particular $\hat{\Sigma}$. Se han planteado estimadores que, de alguna manera, representan una posición intermedia, por ejemplo:

$$\hat{\Sigma}_i(\alpha) = \frac{(1-\alpha)N_i\hat{\Sigma}_i + \alpha N\hat{\Sigma}}{(1-\alpha)N_i + \alpha N} \tag{4.18}$$

esto es, una combinación convexa de los estimadores anteriores con parámetro $0 < \alpha < 1$, elegido para maximizar alguna medida de lo bueno que resulta el clasificador con este nuevo estimador. A estos métodos de estimación, se les denomina shrinkage methods [Campbell 80] [Friedman 89], o métodos de contracción, y pretenden obtener un estimador sesgado pero menos variable. Existen, por supuesto, otras posibilidades de hacer combinaciones de matrices [Ripley 96] [Hoffbeck 96].

Aparte del enfoque *plug-in*, existen otros que también pueden resultar eficientes aunque quizás son menos utilizados en la práctica.

Uno de estos enfoques alternativos es el propuesto por Moran & Murphy [Moran 79], que viene a ser una versión sofisticada de las estrategias, ya comentadas, encaminadas a reducir el sesgo de los estimadores. En este caso, lo que se pretende es proporcionar un estimador no sesgado, directamente de $\log p(x \mid C_i)$, en lugar de tratar de corregir el sesgo de $\hat{\Sigma}_i$ o $\hat{\Sigma}$, como ya vimos. Así, la regla para asignar un dato a la clase 1 utilizando un discriminante lineal, quedaría ahora como:

$$\frac{N-2-n-1}{N-2}\left[(\hat{\mu}_1-\hat{\mu}_2)^T \tilde{\Sigma}^{-1}\left(x-\hat{\bar{\mu}}\right)\right]+\frac{n}{2}\left[\frac{1}{N_1}-\frac{1}{N_2}\right]+\log\frac{P(C_1)}{P(C_2)}>0 \qquad (4.19)$$

donde $\bar{\mu}=\dfrac{\mu_1+\mu_2}{2}$.

Si se compara con la expresión habitual basada en los estimadores de máxima verosimilitud:

$$(\hat{\mu}_1-\hat{\mu}_2)^T \hat{\Sigma}^{-1}\left(x-\hat{\bar{\mu}}\right)+\log\frac{P(C_1)}{P(C_2)}>0 \qquad (4.20)$$

se observan claramente los cambios introducidos. El efecto sobre el discriminante cuadrático es incrementar la varianza efectiva por un factor $N_i/(N_i-n-1)$ con respecto al habitual $N_i/(N_i-1)$, y añadir una constante que depende de N_i.

La otra aproximación al problema, es lo que se conoce como métodos predictivos de estimación de densidades y probabilidades a posteriori paramétricas, y están inspirados en la estadística bayesiana [Aitchison 75] [Geisser 93].

A diferencia de lo que ocurre en los métodos *plug-in*, los parámetros θ se consideran variables aleatorias descritas por una distribución de *prior* $p(\theta)$, y de esta forma se modela la incertidumbre que se tiene acerca de su verdadero valor.

Si se calculan las densidades de clase usando esta aproximación, obtendríamos las probabilidades a posteriori como:

$$\tilde{p}(x \mid C_i) = \int p(x;\theta \mid C_i) p(\theta \mid D) d\theta \,, \qquad \tilde{P}(C_i \mid x) = \frac{P(C_i)\tilde{p}(x \mid C_i)}{\sum_{k=1}^{L} P(C_k)\tilde{p}(x \mid C_k)} \tag{4.21}$$

Calcular estas integrales puede llegar a ser muy complicado, empezando porque hay que suponer una forma para $p(\theta)$. En la práctica, se suelen hacer bastantes aproximaciones y se calculan explícitamente para algunos casos especiales importantes [Ripley 88].

Realmente, esta forma de atacar el problema, no ha tenido demasiada repercusión en la literatura de reconocimiento de patrones. Esto puede deberse a que, para el tipo de familias paramétricas de distribuciones que se manejan usualmente, las diferencias con otros métodos son insignificantes. Sin embargo, si se consideran familias mucho más amplias, pueden llegar a ser importantes.

4.1.3.2 Determinación de Parámetros en Redes Neuronales

Como ya se comentó en el Capítulo 1, los parámetros de una red neuronal se denominan pesos. Centrándonos en el perceptrón multicapa (MLP), que es el tipo de red neuronal más popular, tenemos que esta red implementa una función $R^n \rightarrow M \in R^L$, siendo M acotado; n el número de entradas y L el de salidas. La función concreta queda determinada al fijar los valores de los pesos. Se ha demostrado que, dado un número suficientemente alto de neuronas, un MLP de al menos dos capas ocultas es capaz de representar o aproximar con precisión arbitraria cualquier función no lineal.

Desafortunadamente, no es inmediato lograr la función deseada, ya que la configuración de pesos que hace esto posible es desconocida. Es necesario someter al

MLP a un proceso de entrenamiento en el que mediante un algoritmo de aprendizaje se determinen los pesos adecuados. El ajuste de los pesos se realiza normalmente mediante un proceso de búsqueda de gradiente tratando de minimizar una función objetivo. Dicha función suele ser el error cuadrático (4.2), aunque existen otras posibilidades como la función de error de entropía cruzada [Hopfield 87] [Hampshire 90], dada por:

$$E = -\sum_{j=1}^{N}\sum_{i=1}^{L} t_i^j \log\left(\frac{y_i^j}{t_i^j}\right) \tag{4.22}$$

El procedimiento para calcular el gradiente en una red de tipo MLP se denomina retropropagación (*backpropagation*) y fue inicialmente publicado por Bryson y Ho [Bryson 69] y posteriormente redescubierto varias veces. Se popularizó con la publicación del primer volumen del grupo PDP, [Rumelhart 86]. Se basa en la aplicación de la regla de la cadena en un proceso en dos fases. En una primera se alimentan las entradas para producir en la capa final la salida y el error cuadrático. En una segunda fase, el error es transmitido hacia atrás, calculándose la contribución relativa al mismo de cada elemento hasta llegar a la capa de entrada. Con esta magnitud se deriva el gradiente. El gradiente se emplea directamente en la regla de aprendizaje para determinar las correcciones en los pesos y vuelve a comenzar el ciclo.

La velocidad en el aprendizaje viene fundamentalmente determinada por el coeficiente de aprendizaje η. Este coeficiente puede mantenerse fijo o intentar adaptarlo de manera que se acelere el algoritmo. Es posible, en lugar de utilizar un η fijo, realizar la búsqueda del η^* que produce el mínimo del error cuadrático en la dirección del gradiente. Esta búsqueda es unidimensional, y se realiza iterativamente. En cada iteración el coste en cálculo es una estimación del error cuadrático. Al aplicarse el paso η^* se vuelve a calcular el gradiente y es posible repetir el ciclo con paso η^* hasta que se juzgue conveniente actualizarlo.

Una modificación muy común en la regla de aprendizaje del MLP es la introducción del término de *momento,* $\alpha\left[\Delta w_{ij}(k)\right]$

$$w_{ij}(k+1) = w_{ij}(k) - \eta \left.\frac{\partial \mathbf{E}(\mathbf{w})}{\partial w_{ij}}\right|_{\mathbf{w}(k)} + \alpha\left[\Delta w_{ij}(k)\right]$$

(4.23)

Dicho término acelera el entrenamiento permitiendo escapar de pequeñas variaciones locales en el gradiente, debido a que la nueva dirección a seguir contiene una componente de las direcciones de búsqueda seguidas en el pasado.

Otra técnica ensayada consiste en aplicar los métodos de optimización de gradiente conjugado. Estas técnicas se basan en que es posible conseguir una convergencia más rápida que la del método del gradiente descendente en las cercanías de un mínimo. Dada la suficiente proximidad a un mínimo, cualquier función se puede aproximar por una función cuadrática en ese entorno. Para el caso de la función cuadrática es posible elegir adecuadamente las direcciones de búsqueda de forma que se alcance el mínimo en *n* pasos, siendo *n* la dimensión de la superficie. Las direcciones sucesivas de búsqueda se determinan a partir de las anteriores y se denominan direcciones conjugadas. Es de esperar que si está en las cercanías de un mínimo, el número de pasos necesario para alcanzarlo se reduzca. En este caso, se emplea información del gradiente y de las direcciones anteriores para determinar la nueva dirección conjugada a seguir, mediante una combinación lineal de los mismos.

En el algoritmo de gradiente conjugado, se hace uso implícito de información de segundo orden acerca de la superficie de error, representada por la matriz Hessiana local. Existen una clase de métodos que hacen uso explícito de esta matriz y son los que se denominan métodos Newton.

De forma aproximada, el gradiente en cualquier punto w (conjunto de pesos) vendrá dado por [Bishop 95]:

$$\nabla E = H\left(w - w^*\right) \tag{4.24}$$

siendo ∇E el vector de gradiente local, H la matriz Hessiana y w^* el punto donde se alcanza el mínimo de la función de error, que podrá calcularse entonces como:

$$w^* = w - H^{-1}\nabla E \tag{4.25}$$

Al vector $-H^{-1}\nabla E$ se le conoce como la dirección de Newton o el paso de Newton, y constituye la base para diferentes estrategias de optimización. A diferencia del vector de gradiente local, la dirección de Newton, evaluada en cualquier punto w de una

superficie cuadrática, apunta directamente al mínimo de la función de error. Cuando la aproximación cuadrática no es válida, se hace necesario aplicar la expresión (4.25) iterativamente.

La implementación directa de los métodos Newton es computacionalmente prohibitiva y por ello, en la práctica, suelen utilizarse los denominados métodos *quasi-Newton* que tratan de paliar el excesivo cálculo que implica (4.25).

Finalmente, otro algoritmo importante es el de Levenberg-Marquardt [Levenberg 44] [Marquardt 63], que sólo es aplicable a funciones de error cuadrático.

4.1.4 El Problema de la Dimensionalidad

La dimensionalidad viene dada por el número de características con las que se trabaja. Añadir características significa añadir información, o en el peor de los casos, quedarnos como estábamos. Esto es así teóricamente, porque en la práctica, a menudo, se observa que cuando se aumenta el número de características, el rendimiento del clasificador empeora. A este sorprendente efecto, debido a una excesiva dimensionalidad, se le conoce como *The Curse of Dimensionality*, y se produce por el hecho de trabajar con un conjunto de datos finito. Este efecto es especialmente dramático cuando se utilizan modelos de clasificadores no paramétricos.

Es por eso que la dimensión del espacio de características es un factor muy importante a tener en cuenta, especialmente cuando hablamos de dimensionalidades muy altas. En estos casos, incluso se pierde la intuición geométrica y estadística que nos da la experiencia en espacios tridimensionales [Jiménez 98], influyendo también en las estrategias a emplear para diseñar el clasificador más adecuado.

Por estos motivos, suele resultar ventajoso, la utilización de técnicas que permitan la reducción de la dimensionalidad. Éstas se dividen en dos grandes grupos.

- Técnicas que consisten en hacer combinaciones (normalmente lineales) de características que tengan un buen poder discriminatorio entre clases. De hecho, esto es equivalente a utilizar un discriminante lineal y por eso el discriminante de Fisher constituye un ejemplo de este tipo de técnicas. Aunque sin duda la técnica más extendida es el análisis de componentes principales [Watanabe 69] [Devijver 82], si bien, en principio, no parece que tenga nada que ver con la capacidad de

discriminación. En la figura 4.2 se muestra un ejemplo de lo que se acaba de comentar, pasando de dos a una sola característica. Queda claro que es muy importante elegir adecuadamente la transformación a aplicar, ya que en un caso se obtiene una buena separabilidad entre las clases y en el otro no.

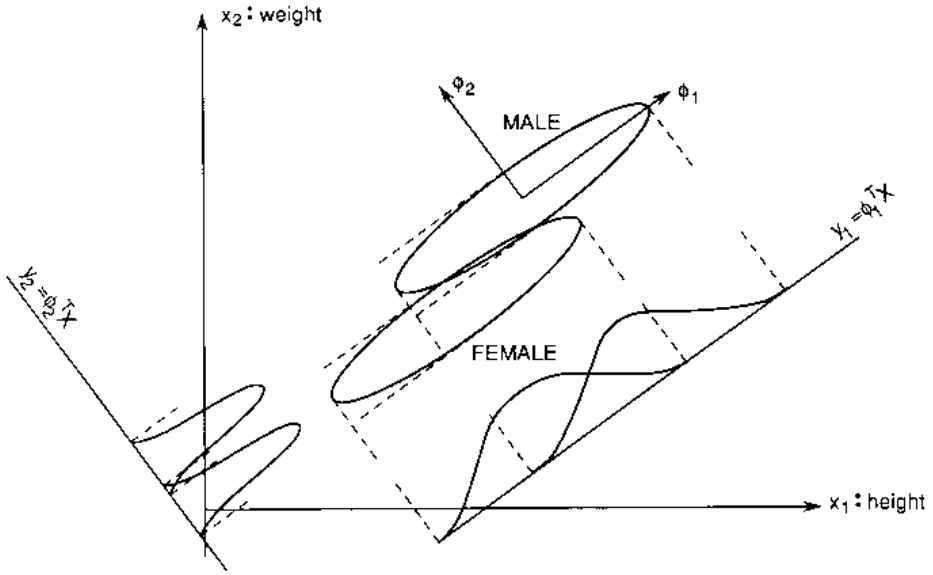

Figura 4.2. Un ejemplo de extracción de características (figura extraída de [Fukunaga 90])

- Técnicas que consisten en eliminar, del conjunto original, características que a través de algún procedimiento, dejen evidencia de su pobre contribución a la discriminación entre clases [Dash 97]. Normalmente de se trata de técnicas iterativas que añaden o quitan una característica de cada vez y entre ellas cabe citar: selección hacia delante y hacia detrás ("Forward and Backward Selection") y métodos del tipo "Branch-and-Bound".

Para la aplicación de estas técnicas, es necesario utilizar algún criterio que permita evaluar la capacidad discriminatoria del conjunto reducido de características. Con este fin, y a efectos de simplificación, se suelen usar medidas de separabilidad entre clases, entre las cuales se encuentra la distancia de Bhattacharyya [Fukunaga 90], que por ser una de las más usadas merece que nos detengamos un poco en ella. Se define para distribuciones normales, como:

$$J_B = \frac{1}{8}(\mu_2 - \mu_1)^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} \log \left(\frac{\left| \frac{\Sigma_1 + \Sigma_2}{2} \right|}{\sqrt{|\Sigma_1||\Sigma_2|}} \right) \qquad (4.26)$$

donde, los M_i representan los vectores de medias y los Σ_i, las matrices de covarianza.

Como puede verse, consta de dos términos, el primero que da la separabilidad entre clases debido a la diferencia de las medias, y el segundo que refleja la separación introducida por la diferencia de covarianzas. Aunque se haya definido para distribuciones de normales, parece una medida razonable aplicable a otro tipo de distribuciones, con la ventaja añadida de que $\sqrt{P(C_1)P(C_2)}e^{-J_B}$ resulta ser una cota superior del error de Bayes.

Hay que tener en cuenta que todas estas técnicas, suponen normalmente, una perdida de información, que de llegar a ser demasiado importante, podría no compensar la reducción lograda en la dimensionalidad.

4.1.5 Evaluación de las Prestaciones de un Clasificador

En el apartado 4.1.2, expresamos de forma cuantitativa, la influencia que, sobre el cálculo del error cuadrático de clasificación, tiene, el disponer de un conjunto finito de datos de entrenamiento. Esta influencia quedaba entonces en función de dos términos de bias y varianza. Sin embargo, se asumió que para el cálculo efectivo del error se disponía de infinitas muestras (se integraba sobre todos los valores de x). Esto, obviamente, no representa para nada la situación real, en la cual, no suele quedar otro remedio, que proceder a contar cuantos de los datos disponibles han sido asignados a cada clase. Por lo tanto, aparte del sesgo y varianza introducidos como consecuencia del proceso de entrenamiento, hay también que tener en cuenta esta circunstancia.

Existen varias formas de organizar los datos para el cálculo del error. La más directa, quizás, consiste en utilizar los mismos datos que en el entrenamiento. A este procedimiento se le llama método de resustitución (*Resubstitution Method*). El problema con este método es que aparte de añadir más varianza al error, también contribuye al sesgo y de hecho da una estimación optimista del error.

Más fiable resulta el procedimiento conocido como método *Holdout* que consiste en dividir el conjunto de datos en dos subconjuntos, uno para el entrenamiento y otro para test o validación. Como ahora tenemos dos conjuntos de datos independientes, no habrá contribución al sesgo (viniendo enteramente del proceso de entrenamiento), y si que seguirá habiendo contribución a la varianza. Cuando los datos se generan artificialmente, éste parece un procedimiento ideal, pero en la práctica supone dividir, aún más, un conjunto de datos, de por sí reducido.. Además está la cuestión, no trivial, de cómo hacer la división, ya que, debemos las distribuciones de ambos subconjuntos deberían ser los más parecidas posibles, e incluso, no está claro cuantos datos deben de recaer en cada uno de ellos.

Por todo esto, puede resultar más indicado, aplicar otro procedimiento denominado método *Leave-One-Out*. En este método, se excluye un dato, se entrena el clasificador con los $N-1$ restantes, y el dato excluido es utilizado para validar el clasificador. Esta operación es repetida N veces. Como cada dato de test, se excluye del conjunto de entrenamiento, queda garantizada la independencia. Además se utilizan de forma efectiva todos los datos, tanto para entrenar como para validar, por eso a este tipo de métodos se les denomina de validación cruzada (*Cross-Validation*). También, no hay necesidad de precocuparse por diferencias entre las distribuciones de los datos de entrenamiento y test. Quizás el inconveniente más serio que presenta sea el hecho de tener que entrenar N clasificadores, lo cual, puede suponer un tiempo de computación considerable. Afortunadamente, para ciertos clasificadores, como los discriminantes lineal y cuadrático, este tiempo puede ser prácticamente igual al equivalente a entrenar un solo clasificador [Fukunaga 90].

Finalmente, cabe citar otros métodos que también se utilizan en la estimación del error aunque basados en una filosofía un tanto diferente, por ejemplo, los métodos *Bootstrap* [Efron 82]o *Jackknife* [Quenouille 49].

Hasta ahora, nos hemos referido siempre a una única medida de las prestaciones de un clasificador. En ciertos dominios, como la Medicina, puede ser de gran importancia, ponderar adecuadamente, este error entre las diferentes clases. Así, y a modo de ejemplo, no supone lo mismo diagnosticar a un paciente enfermo como sano, que a uno sano como enfermo.

Siguiendo en el mismo dominio, aún cuando el error obtenido sea muy bajo, puede resultar inaceptable viniendo de un sistema tipo “caja negra”. Por eso, cierto tipos

de clasificadores, que no resultan tan eficientes, en términos del error de clasificación, pueden ser preferibles debido a su capacidad explicativa.

4.2 El Problema del Diseño en General

La resolución de una tarea de diseño es una actividad compleja que implica gran número de subtareas y métodos para resolver cada una de ellas. La estructura de tareas de este problema ha sido objeto de investigación en metodologías orientadas a tareas para Sistemas Basados en el Conocimiento [Chandrasekaran 90]. En particular, nos centraremos en aquellos métodos de resolución que utilicen intensivamente el conocimiento explícito disponible, sobre el problema particular a tratar. Quedan fuera de este enfoque, por lo tanto, aquellos problemas de diseño, bien condicionados, en los que la solución podría obtenerse, por ejemplo, tras resolver un sistema de ecuaciones.

De forma muy general, se puede definir el problema del diseño como una búsqueda en un espacio de objetos (posibles diseños) que deben satisfacer unos requisitos y no violar ciertas restricciones.

En principio, como solución, cabría plantearse una búsqueda exhaustiva, de tal forma que se fueran probando todos los candidatos hasta dar con el diseño adecuado. Esta aproximación al problema sólo es válida para pequeños espacios de soluciones, ya que de lo contrario, el proceso resultaría excesivamente costoso e incluso impracticable. Por este motivo, las diferentes estrategias suelen ir encaminadas a restringir lo más posible este espacio de posibles soluciones.

Los métodos del tipo Proponer-Criticar-Modificar [Chandrasekaran 90] constituyen una opción muy popular y, habitualmente, eficiente, de resolver el problema. Se trata de métodos muy simples, con muchas variantes, y cuya estructura básica es la siguiente:

1. Si el diseño no se ha completado, proponer una nueva extensión de diseño (por ejemplo, añadir una pieza). De lo contrario, salir.
2. Verificar el diseño hasta este punto. Si satisface los requisitos y no viola ninguna restricción, volver al paso 1. De lo contrario, ir al paso 3.
3. Criticar el diseño hasta este punto y generar un lista ordenada de posibles acciones a realizar para revisar el diseño.

4. Modificar el diseño de acuerdo con las acciones propuestas y volver al paso 2.

4.3 El Diseño de un Clasificador Abordado desde la Perspectiva de un Problema de Diseño Genérico

Volviendo al tema que nos ocupa, hemos visto que se trata de un problema muy complejo, en el que la combinación de los diferentes tipos de clasificadores, el número de parámetros a considerar, los métodos de estimación de estos parámetros, etc..., pueden dar lugar a conjunto muy grande de posibles diseños, entre los cuales tenemos que seleccionar el más adecuado. Nos encontramos, por lo tanto, en una situación similar a la planteada en la sección anterior, que trataremos de solventar planteando una variante del esquema Proponer-Criticar-Modificar.

Este tipo de métodos se basan en la disponibilidad de conocimiento que permita llevar a cabo los diferentes pasos descritos con anterioridad. En este dominio, resulta bastante complicado disponer de este tipo de conocimiento. A pesar de todo, la experiencia adquirida a través de la realización de este trabajo y la recabada en publicaciones diversas en el área del reconocimiento de patrones, nos han llevado a plantear una serie de heurísticas con las que trataremos de resolver el problema satisfactoriamente. Estas heurísticas han sido integradas en un Sistema Basado en el Conocimiento, que se ha construido utilizando la metodología CommonKADS, siguiendo las pautas descritas en capítulos anteriores. Así, empezaremos hablando del Modelo de Conocimiento para luego pasar a su implementación.

4.3.1 Modelo de Conocimiento del Problema de Diseño de un Clasificador

El Modelo de Conocimiento completo, escrito en el lenguaje CML, se encuentra en el apéndice. En esta sección, se presentarán de forma más esquemática los tres tipos de conocimiento que nos podemos encontrar: el Conocimiento de Dominio, el Conocimiento de Inferencia y el Conocimiento de Tarea.

La figura 4.3 muestra los dos tipos de Conocimiento de Tarea: Tarea y Método de Tarea, y la descomposición en Inferencias.

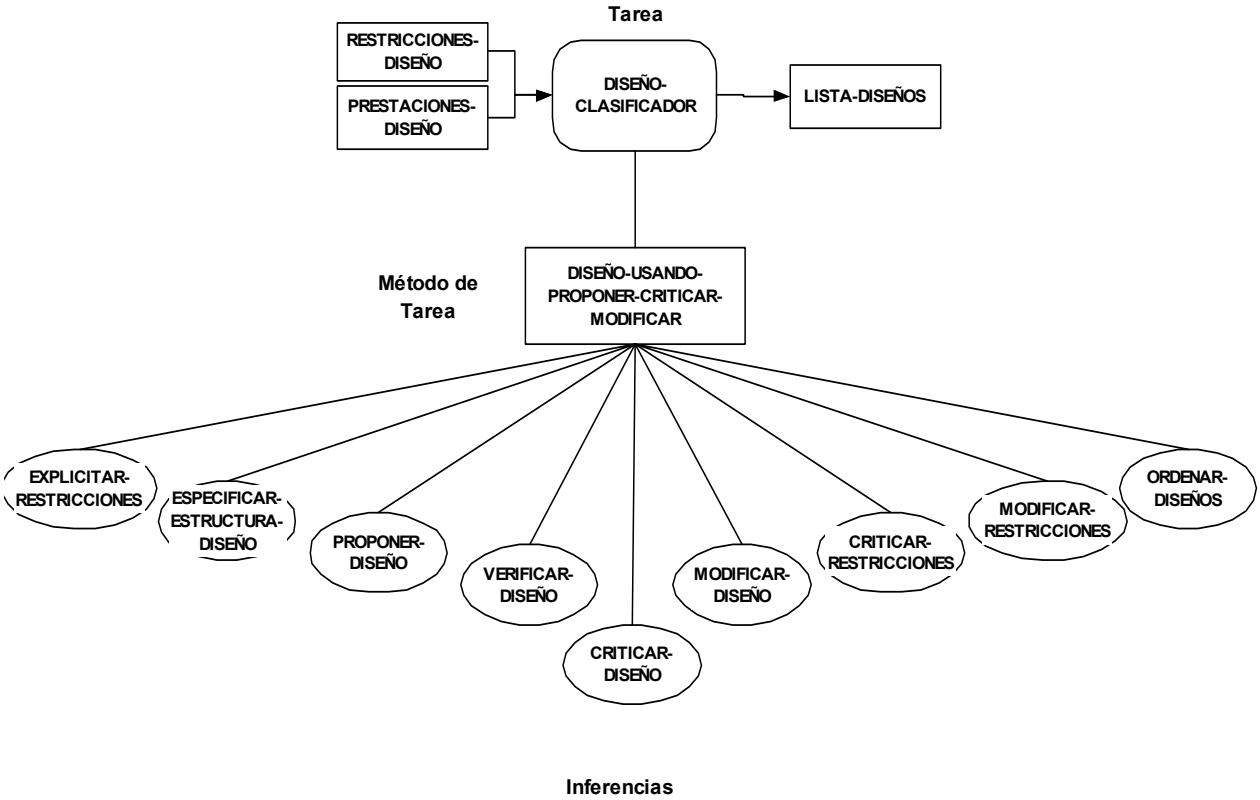

Figura 4.3. Tarea, Método de Tarea y descomposición en Inferencias para el problema del diseño de un clasificador

Se observa que a las inferencias propias del método Proponer-Criticar-Modificar, se han añadido otras, que serán descritas posteriormente. Es importante destacar, que no todas las inferencias se apoyan en conocimiento explícito, contenido en las diferentes bases de conocimiento. En algunas, la relación entre la entrada y la salida viene dada por algoritmos que realizan cálculos más o menos complejos.

4.3.1.1 El Rol Diseño y el Concepto Clasificador

En este punto, y ya que hablamos de la tarea de más alto nivel, que es el diseño del clasificador, parece lógico empezar por definir que debemos entender por diseño.

El Rol de Diseño lo juega el Concepto Clasificador, definido, como se muestra en la figura 4.4.

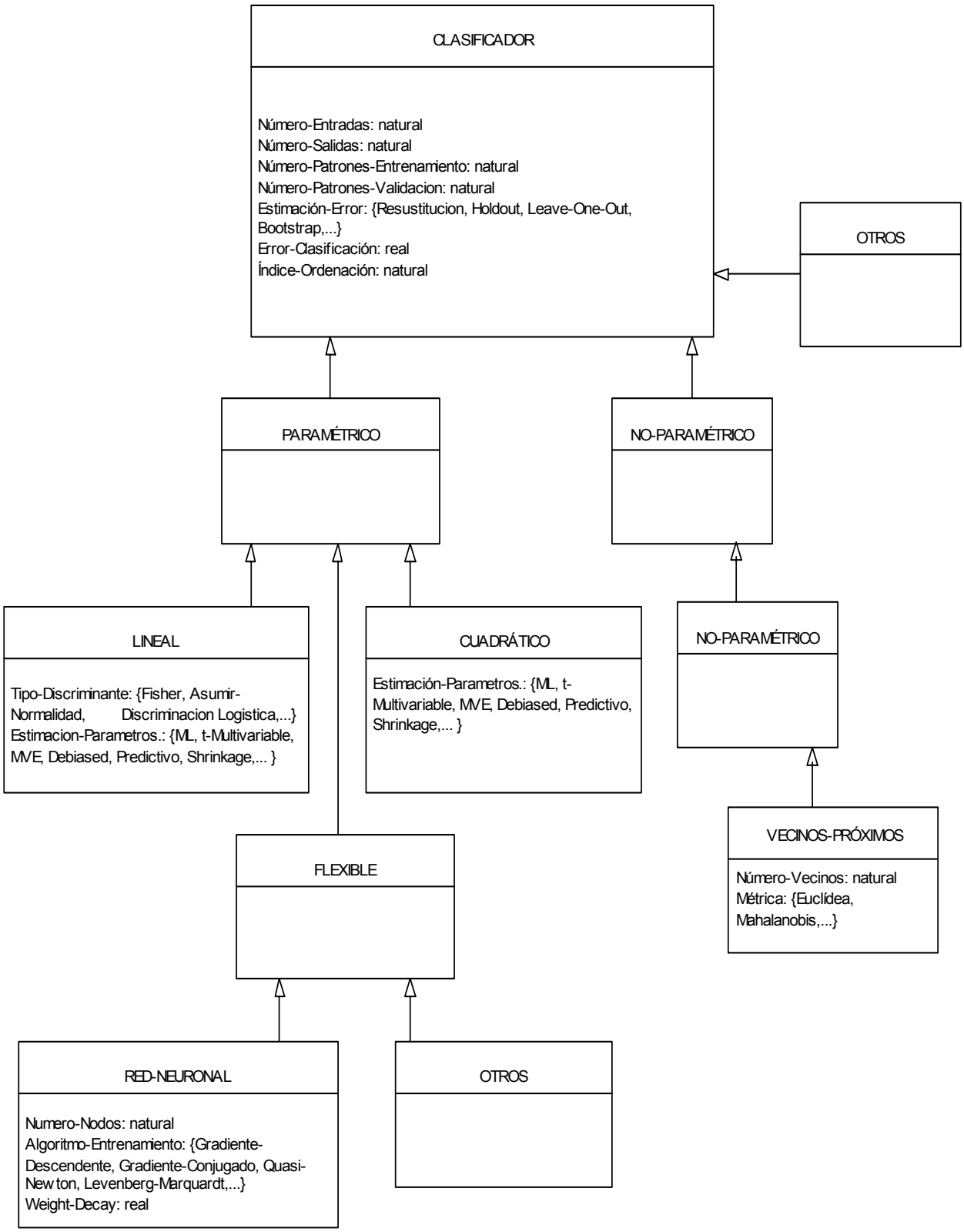

Figura 4.4. El Concepto Clasificador y los conceptos directamente relacionados con éste

Se puede observar, que se trata, en realidad, de varios conceptos, organizados jerárquicamente, agrupando los clasificadores por afinidad en las cuestiones

relacionadas con el diseño de cada uno de ellos. De hecho, esta es la forma habitual de agrupar los diferentes tipos de clasificadores, y puede encontrarse en numerosos libros y artículos relacionados con el tema [Ripley 96].

Como tenemos una relación entre conceptos de la forma Subtipo de (SUBTYPE-OF), todos los conceptos heredan los atributos del concepto central Clasificador, y a estos hay que añadir, los particulares de cada concepto. En definitiva, un diseño vendrá dado por este tipo de estructura, con todos sus atributos fijados a un determinado valor, esto es, un modelo de clasificador, entrenado y validado, por ejemplo:

```
CONCEPT Clasificador;
   DESCRIPTION: "Objeto que describe el clasificador a través de una serie de parámetros generales";
   ATTRIBUTES:
      Número-Entradas: 5;     /*dimensionalidad*/
      Número-Salidas: 2;      /*nº de clases*/
      Priors-Clases: {1/2,1/2};          /*si no se especifican, se calculan a partir de los patrones de entrada*/
      Número-Patrones-Entrenamiento: {50, 50};      /*en cada clase*/
      Número-Patrones-Validación: {50, 50};  /*en cada clase*/
      Estimación-Error: Holdout;              /*método de estimación del error*/
      Error-Clasificación: 0.9;          /*estimación del error*/
      Índice-Ordenación: 0;     /*índice de ordenación*/
END CONCEPT Clasificador;

CONCEPT Lineal;
   DESCRIPTION: "Clase de discriminantes lineales";
   SUBTYPE-OF: Paramétrico;
   ATTRIBUTES:
      Tipo-Discriminante: Fisher;             /*tipo de discriminante lineal*/
      Estimación-Parámetros: ML;     /*método de estimación de parámetros*/
END CONCEPT Lineal;
```

Los atributos de estos dos conceptos definen un diseño, que posteriormente habrá que comprobar, si es o no válido. Como aclaración hay que decir que el atributo Número-Entradas coincide con la dimensionalidad del problema (n); el atributo Número-Salidas, con el número de clases del problema (L); y los atributos que hacen referencia al número de patrones son listas , cada una de ellas con una lista de valores ordenada por clases. Además, se han utilizado algunos nombres clave para designar, sobre todo, a

diferentes estimadores. Así, en este caso, ML se refiere a *Maximum Likelihood*. Finalmente, el atributo Índice-Ordenación, inicializado a cero, se utilizará con el fin de ordenar de forma definitiva los diseños de clasificadores válidos. Esta ordenación se hará de acuerdo con ciertos criterios que serán explicados más adelante.

4.3.1.2 Estructura de Inferencias y su Relación con los Elementos del Dominio

En la figura 4.5, se muestra la estructura de inferencias para este problema.

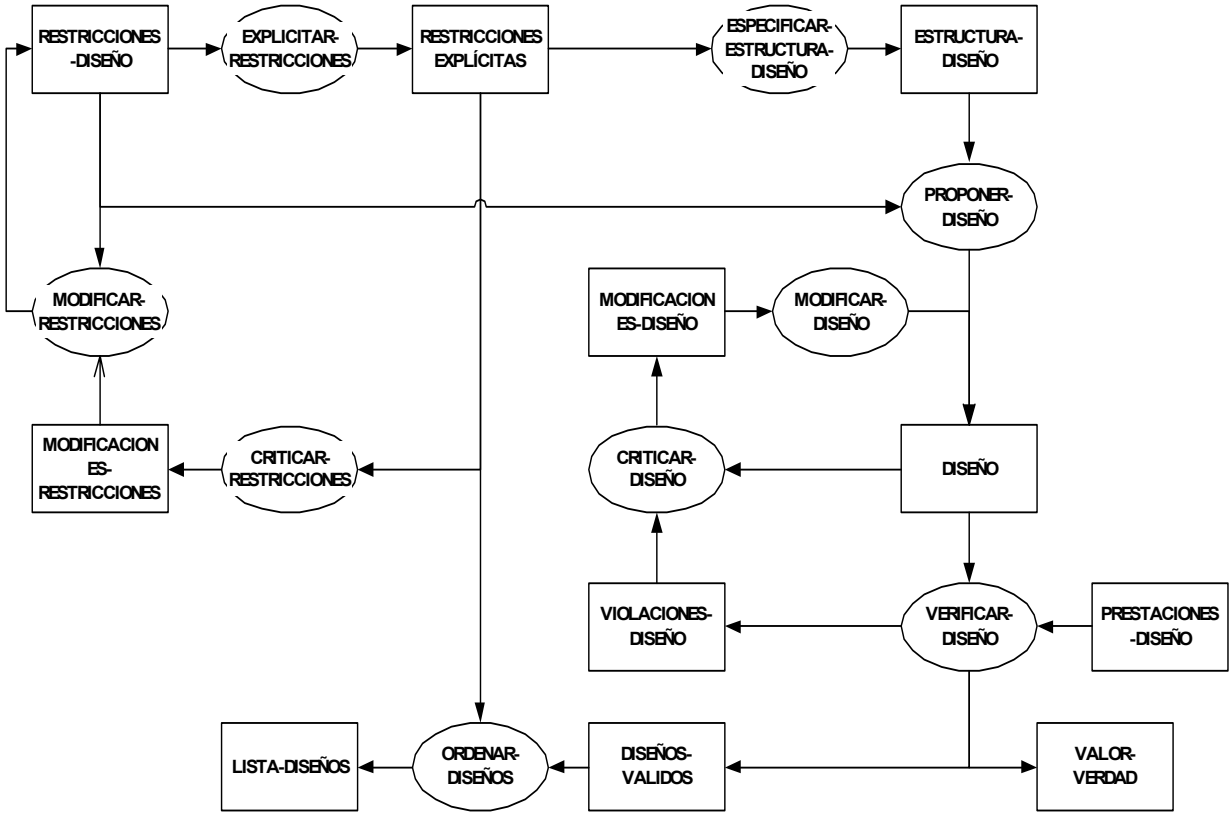

Figura 4.5. Estructura de Inferencias para el problema del diseño de un clasificador

Pasamos ahora a describir cada una de las inferencias y su relación con el dominio a través de los diferentes roles.

Inferencia: Explicitar-Restricciones

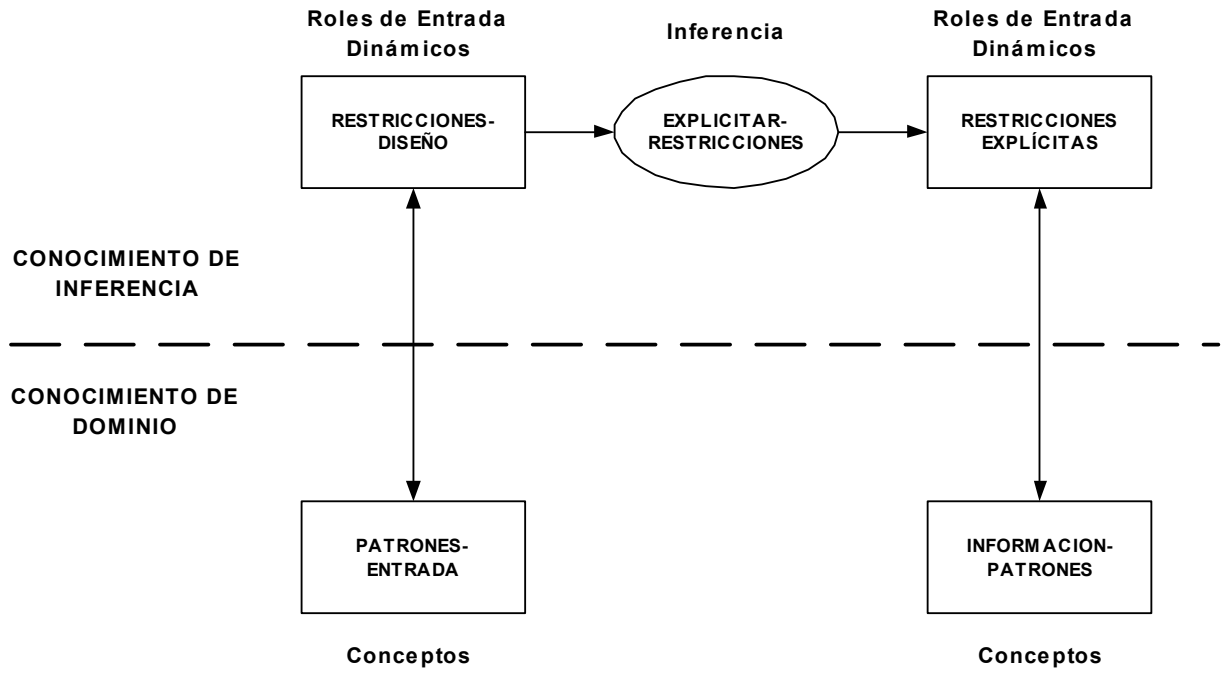

Figura 4.6. Inferencia Explicitar-Restricciones y su relación con el dominio

Objetivo: extraer información de los patrones de entrada y hacerla explícita.

El Rol de Restricciones-Diseño lo juega el concepto Patrones-Entrada, ya que en estos, se encuentra, de forma implícita, la información que condiciona y, en definitiva, restringe, el diseño del clasificador. Por otro lado, el Rol de Restricciones-Explícitas, viene dado por el concepto Información-Patrones, que como su propio nombre indica, guarda en sus atributos, información obtenida a partir de los datos de entrada.

Parte de esta información es trivial de obtener, como puede ser el número de clases. Sin embargo, otro tipo de información, precisa de algún cálculo bastante más complejo. En este sentido, se han incluido tres procedimientos para la verificación de la normalidad; se calcula la distancia de Bhattacharyya, distinguiendo entre sus dos términos; se calculan dos términos de sesgo para el discriminante lineal y el cuadrático; y se calcula la dimensionalidad intrínseca de los datos. Para hacernos una idea más clara de la información recabada, mostramos un ejemplo de cómo podría quedar el Concepto Información-Patrones después de que se haya llevado a cabo la inferencia.

CONCEPT Información-Patrones;

```
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {100, 100};  /*en cada clase*/
      Número-Clases: 2;     /*nº de clases*/
      Número-Características: 3;                /*nº de características*/
      Test1-Normalidad-Mono: {0.7,0.8};              /*probabilidades posteriori en cada
clase*/
      Test2-Normalidad-Mono: {0.6,0.9};              /*probabilidades posteriori en cada
clase*/
      Test1-Normalidad-Multi: {0.3,0.45};      /*valores P en cada clase*/
      Distancia-Bhattacharyya: 0.82; /*para dos clases*/
      Distancia-Bhattacharyya-1: 0.41;         /*para dos clases*/
      Distancia-Bhattacharyya-2: 0.41;         /*para dos clases*/
      Sesgo-Lineal: 2.5; /*para dos clases*/
      Sesgo-Cuadrático: 2.5;    /*para dos clases*/
      Dimensionalidad-Intrínseca: 5;
      Reducción-Dimensionalidad: ninguno    /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

De la distancia de Bhattacharyya, hemos hablado en el apartado 4.1.4. En lo que respecta a los tests de normalidad, se han utilizado los dos procedimientos desarrollados en el capítulo 2, para verificar la normalidad monodimensional, y un test adicional de normalidad multivariable desarrollado por Doornik y Hansen [Doornik 94], aunque existen muchos más tests de este tipo [Romeu 93]. Las razones por la que hemos escogido éste en particular, son básicamente dos. Por un lado, en su artículo se precisan todos los pasos necesarios para su implementación, y por otro, presentan una serie de pruebas de potencia que parecen bastante satisfactorias. Reseñar también, que aunque los procedimientos desarrollados en el capítulo 3 no son aplicables directamente a distribuciones multimensionales, resultan de gran utilidad (a veces incluso más que los tests multivariable), dado que, la evidencia de no normalidad en alguna dimensión es condición suficiente para descartar la normalidad conjunta. Sobre todo, no hay que olvidar que, en los tests desarrollados en este trabajo, podemos expresar las conclusiones en forma de probabilidades a posteriori (como ya se demostró en el capítulo 3), mientras que en los tests convencionales tendremos que utilizar el valor P. Añadir, que por razones de simplicidad, en los atributos Test1-Normalidad-Mono y Test2-Normalidad-Mono, sólo se guarda el valor correspondiente a la componente de cada clase con probabilidad a posteriori, de ser normal, más pequeña. Así, si se acepta

que esta componente sigue una distribución normal, automáticamente estaremos aceptando la normalidad del resto (aunque no necesariamente la conjunta).

Los atributos Sesgo-Lineal y Sesgo-Cuadrático, se refieren a dos términos de sesgo, cuya deducción aparece en [Fukunaga 90], y que dan cuenta del sesgo promedio que se introduce en el cálculo del error de clasificación como consecuencia de utilizar un conjunto de datos de tamaño finito para estimar los parámetros de los discriminantes lineal y cuadrático. Las expresiones concretas utilizadas son las siguientes:

Sesgo-Lineal: $$\frac{\frac{1}{2\sqrt{2\pi M^T M}} e^{-M^T M/8}\left[\left(1+\frac{M^T M}{4}\right)n-1\right]}{N}$$

Sesgo-Cuadrático:

$$\frac{\frac{1}{4\sqrt{2\pi M^T M}} e^{-M^T M/8}\left[n^2+\left(1+\frac{M^T M}{2}\right)n+\left\{\frac{\left(M^T M\right)^2}{16}-\frac{M^T M}{2}-1\right\}\right]}{N}$$

donde se puede observar la dependencia con el número de datos empleados en la estimación de los parámetros N, la dimensionalidad n, y la separabilidad entre las clases $M^T M$, con $M = \|\mu_2 - \mu_1\|$.

Es importante tener en cuenta, que al igual que ocurre con la distancia de Bhattacharyya, estas medidas se definen sólo para dos clases. Para que resulten de utilidad en el caso más general, lo que se ha hecho es considerar las clases dos a dos, contemplando todas las posibilidades, y quedarnos con el peor de los resultados encontrados. Esto significa que para la distancia de Bhattacharyya se tomará la menor de las medidas (caso más difícil en cuanto a separabilidad), y para los términos de sesgo nos quedaremos con los mayores (más sesgo y por lo tanto peor estimación del error).

La dimensionalidad intrínseca de un conjunto de datos puede definirse como el mínimo número de parámetros requeridos para dar cuenta de las propiedades observadas de los mismos. Geométricamente, vendría a ser la dimensión de la hipersuperficie conteniendo los datos.

Procedimientos como el cálculo de componentes principales, sirven para estimar esta propiedad, pero están limitados al basarse en transformaciones lineales. Por este motivo, hemos utilizado el procedimiento más general propuesto por Fukunaga

[Fukunaga 90], relacionado con la estimación de densidades de probabilidad utilizando técnicas de vecinos próximos.

Finalmente, el atributo Reducción-Dimensionalidad, hace referencia a los procedimientos, ya comentados, para reducir la dimensionalidad de los patrones de entrada. Evidentemente, esto no siempre es necesario, y por eso, en el ejemplo, aparece con el valor ninguno.

Inferencia: Especificar-Estructura-Diseño

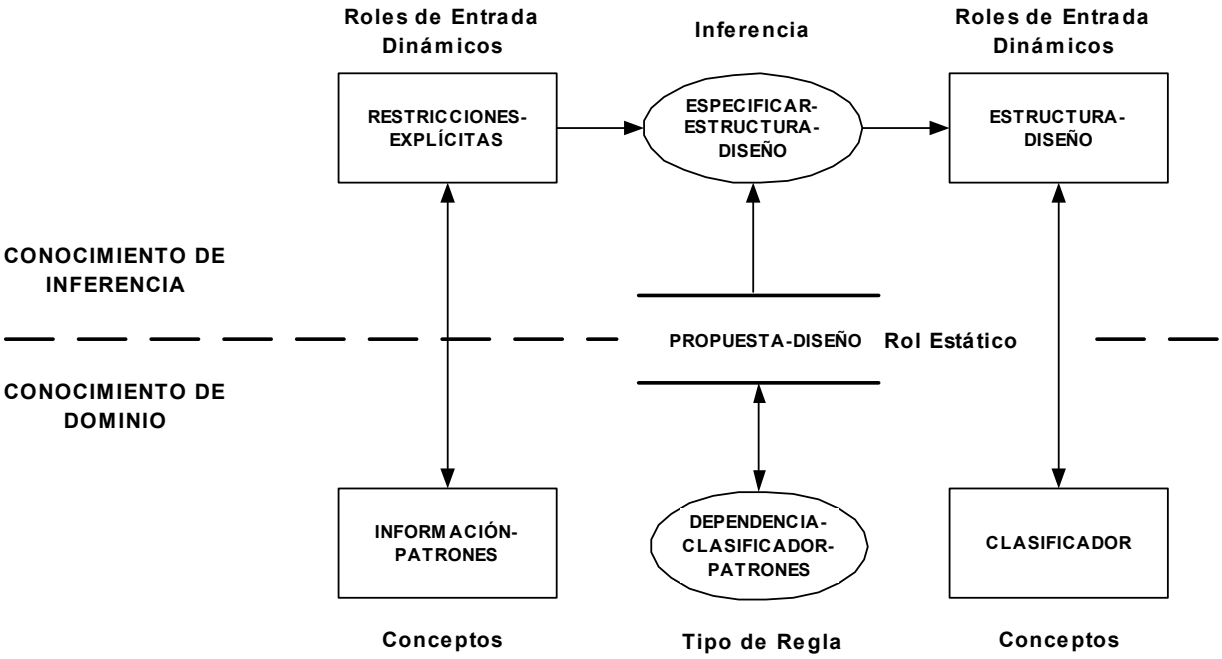

Figura 4.7. Inferencia Especificar-Estructura-Diseño y su relación con el dominio

Objetivo: utilizar la información extraída de los patrones de entrada para especificar un modelo de clasificador y prepararlo para el entrenamiento y validación.

Es una de las inferencias más importantes porque es la que se encarga de hacer las propuestas de los modelos de clasificadores que serán, posteriormente, entrenados y validados. Aunque el Rol Estructura-Diseño se corresponde también con el Concepto

Clasificador, la diferencia con Diseño, es que en este punto, todos los atributos de Clasificador quedan especificados, excepto Error-Clasificacion, ya que todavía no se ha entrenado ni validado el modelo propuesto. Especificar una estructura de diseño no implica sólo seleccionar un tipo de clasificador, sino que normalmente será necesario, también, elegir un método de estimación de parámetros, o un método de estimación del error,...

Para tomar este tipo de decisiones, la inferencia se apoya en el conocimiento expresado en forma de reglas que conforman sus bases de conocimiento asociadas. Se intentará, en la medida de lo posible, justificar las reglas utilizadas, si bien, en un dominio como éste, resulta bastante complicado encontrar argumentos realmente concluyentes.

La Base de Conocimiento quedaría, entonces, de la siguiente manera:

KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores;
USES: Dependencia-Clasificador-Patrones **FROM** Esquema-Dominio-Diseño-Clasificador;
EXPRESSIONS:
Información-Patrones.Número-Clases >= 2 AND Información-Patrones.Distancia-Bhattacharyya-1 > 0.25 AND Información.Patrones.Sesgo-Lineal < 0.1
SUGIERE
Lineal.Tipo-Discriminante = Fisher AND Lineal.Estimación-Parámetros = ML AND Lineal.Estimación-Error = Leave-One-Out;

Información-Patrones.Número-Clases >= 2 AND Información-Patrones.Distancia-Bhattacharyya-2 > 0.25 AND Información-Patrones.Sesgo-Cuadrático < 0.1
SUGIERE
Cuadrático.Estimación-Parámetros = ML AND Cuadrático.Estimación-Error = Leave-One-Out;

Información-Patrones.Número-Clases >= 2
SUGIERE
Red-Neuronal.Número-Nodos = Información-Patrones.Número-Características AND Red-Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-Neuronal.Weight-Decay = 0 AND Red-Neuronal.Estimación-Error = Leave-One-Out;

Información-Patrones.Número-Clases >= 2
SUGIERE
Red-Neuronal.Número-Nodos = 2*Información-Patrones.Número-Características AND Red-Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-

```
Neuronal.Weight-Decay = 0 AND Red-Neuronal.Estimación-Error = Leave-
One-Out;

        Información-Patrones.Número-Clases >= 2
        SUGIERE
        Red-Neuronal.Número-Nodos = 3*Información-Patrones.Número-Características
AND Red-        Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-
Neuronal.Weight-Decay = 0 AND Red-Neuronal.Estimación-Error = Leave-
One-Out;

        Información-Patrones.Número-Clases >= 2
        SUGIERE
        Vecinos-Próximos.Número-Vecinos = 1 AND Vecinos-Próximos.Métrica = Euclídea
AND        Vecinos-Próximos.Estimación-Error = Leave-One-Out;
END KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores;
```

La primera cuestión importante a resaltar, es que estas reglas se han formulado con la idea de descartar modelos de clasificadores que no sean adecuados para el problema, utilizando para ello, la información que se ha extraído de los patrones de entrada. En cualquier caso, y como ya se ha comentado, en este dominio no existen reglas definitivas, lo cual complica notablemente las cosas.

Se puede observar que se proponen cuatro tipos de clasificadores: discriminante lineal, discriminante cuadrático, redes neuronales y clasificador de vecinos próximos.

Como las propiedades de los discriminantes lineal y cuadrático son bien conocidas, resulta más fácil rellenar los antecedentes de las reglas y así descartar su utilización, si no se alcanzan unos mínimos que garanticen su efectividad. En concreto, se ha tenido en cuenta la separabilidad de las clases (medida por los dos términos de la distancia de Bhattacharyya) y los valores de Sesgo-Lineal y Sesgo-Cuadrático en el concepto Información-Patrones. En cualquier caso, hay que decir que los números concretos que figuran en las reglas anteriores son tentativos y susceptibles de ser modificados si se comprueba, a través de la experiencia, que no son las más adecuados. En el consecuente se sugiere el método más sencillo de estimación de parámetros (*Maximum Likelihood*).

En lo que se refiere al resto de clasificadores, el antecedente de sus reglas se cumplirá trivialmente y, por lo tanto, siempre se propondrán como posibles estructuras de diseño.

Para las redes neuronales, se sugieren modelos con una capa oculta conteniendo n, $2n$, y $3n$ neuronas, tratando de garantizar que la red es lo suficientemente compleja como para discriminar problemas bastante difíciles. De hecho, puede demostrarse que con $3n$ neuronas en la capa oculta es posible generar una superficie de decisión esférica que encierre completamente a los patrones de una clase [Hush 89]. Por otro lado, la técnica de regularización *Weight decay* [Hinton 86] que reduce el número de pesos de forma efectiva durante el entrenamiento, depende de un parámetro λ que se fija a cero en estas reglas. El método de entrenamiento propuesto es el *quasi-Newton*.

En el clasificador de vecinos próximos se sugiere el caso más simple con un solo vecino (que suele funcionar bastante bien en la práctica) y métrica euclídea.
Sea cual sea el clasificador propuesto, siempre se calcula el error utilizando *Leave-One-Out*. Esto puede suponer una enorme cantidad de cálculo, si bien, no es una cuestión que nos preocupe en principio.

Inferencia: Proponer-Diseño

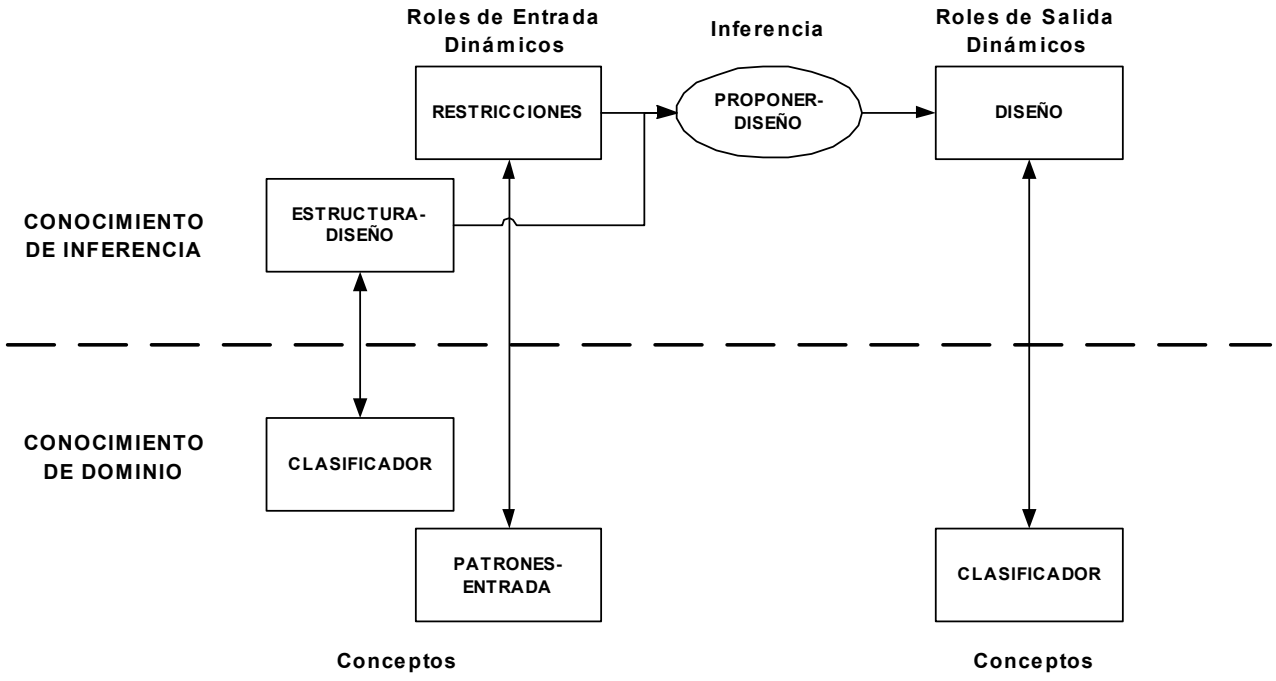

Figura 4.8. Inferencia Proponer-Diseño y su relación con el dominio

Objetivo: entrenar y validar el modelo de clasificador propuesto utilizando los métodos indicados en la estructura de diseño.

Utilizando los patrones de entrada y siguiendo los procedimientos sugeridos, se obtiene un clasificador entrenado y validado como el descrito en el apartado 4.3.1.1.

Inferencia: Verificar-Diseño

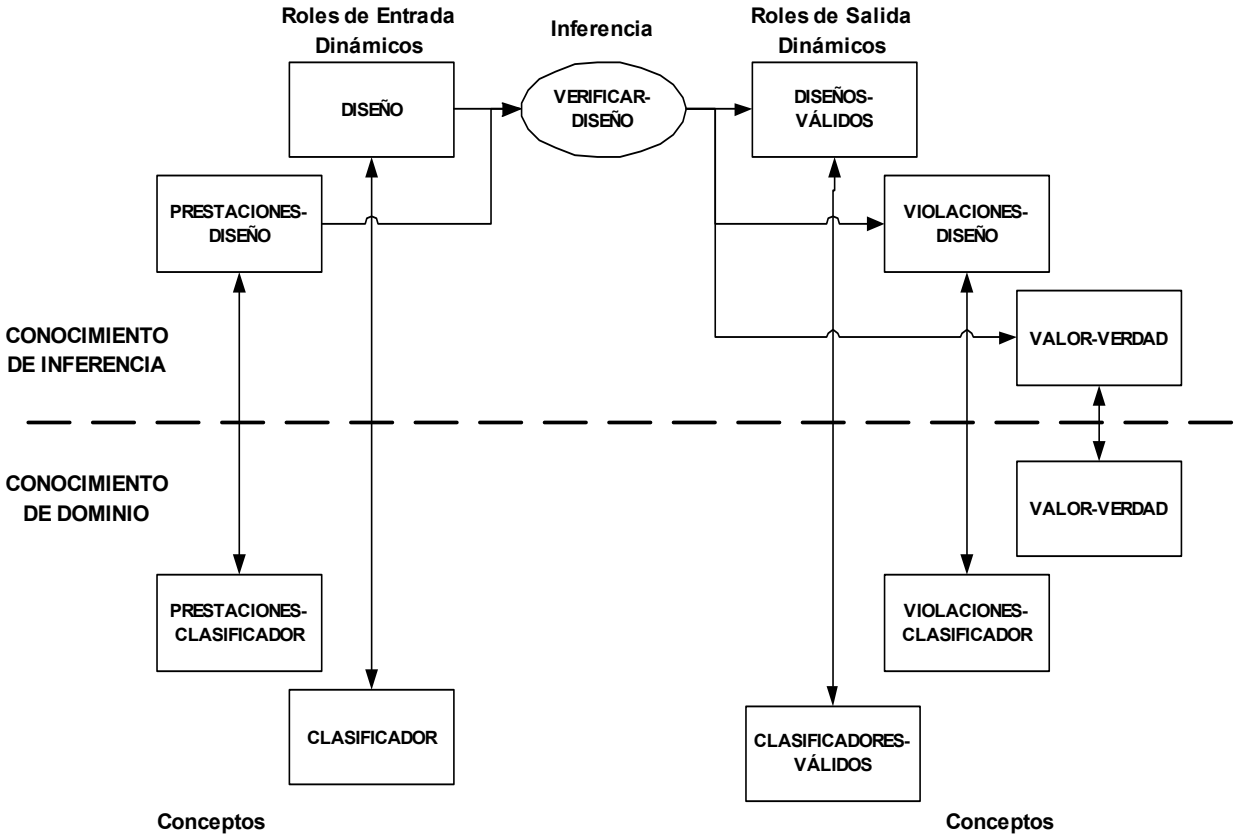

Figura 4.9. Inferencia Verificar-Diseño y su relación con el dominio

Objetivo: verificar si el clasificador propuesto alcanza las prestaciones deseadas.

Para este problema, el Rol de Prestaciones-Diseño vendrá dado, normalmente, por una especificación de error tolerable en el concepto Prestaciones-Clasificador, que en el caso de no poder ser satisfecha por ninguno de los posibles diseños, será necesario relajar. Por su parte, el Rol Violaciones-Diseño apunta al concepto Violaciones-Clasificador, e indica las condiciones no satisfechas. Si bien, cuando hay una sola

condición, este Rol puede resultar un tanto redundante, se ha incluido por generalidad, ya que en ese caso, bastaría con el Rol Valor-Verdad, que simplemente indica si el diseño es válido o no. En el concepto Clasificadores-Válidos se van acumulando los modelos que cumplen con las prestaciones exigidas.

Inferencia: Criticar-Diseño

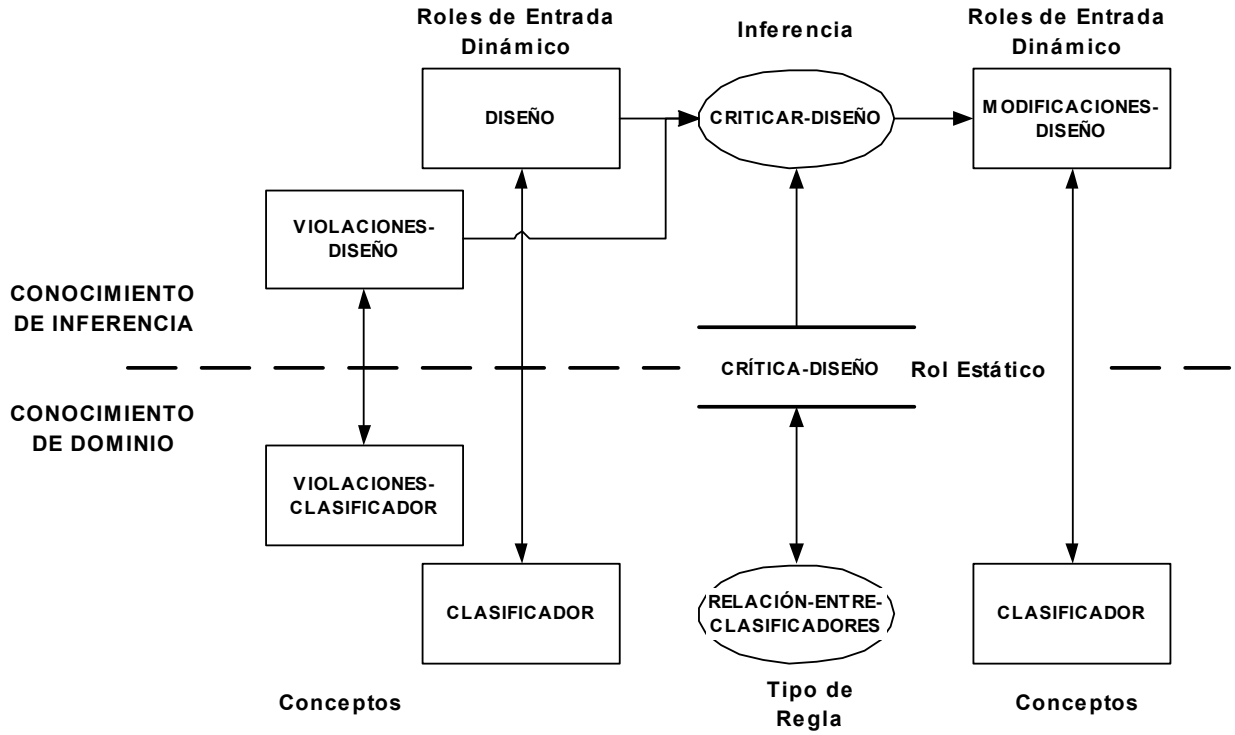

Figura 4.10. Inferencia Criticar-Diseño y su relación con el dominio

Objetivo: criticar el diseño, sugiriendo posibles modificaciones del mismo.

Esta es otra inferencia especialmente importante, porque trata de cambiar algún aspecto del proceso de diseño con el fin de alcanzar las prestaciones exigidas.

Las modificaciones se traducen en los nuevos valores que toman los atributos del concepto Clasificador, y van encaminadas a cambiar, por ejemplo, el estimador de parámetros utilizado en algún discriminante estadístico, o el algoritmo de entrenamiento de una red neuronal(ver apartados 4.1.3.1 y 4.1.3.2).

Las reglas contenidas en la Base de Conocimiento Críticas-Modelos Clasificadores, son comentadas a continuación:

```
KNOWLEDGE-BASE Críticas-Modelos-Clasificadores;
   USES: Relación-Entre-Clasificadores FROM Esquema-Dominio-Diseño-Clasificador;
   EXPRESSIONS:
      Lineal.Estimación-Parámetros = ML
      CAMBIAR-A
      Lineal.Estimación-Parámetros = t-Multivariable;

      Lineal.Estimación-Parámetros = ML
      CAMBIAR-A
      Lineal.Estimación-Parámetros = MVE;

      Lineal.Estimación-Parámetros = ML
      CAMBIAR-A
      Lineal.Estimación-Parámetros = Debiased;

      Lineal.Estimación-Parámetros = ML
      CAMBIAR-A
      Lineal.Estimación-Parámetros = Predictivo;

      Cuadrático.Estimación-Parámetros = ML
      CAMBIAR-A
      Cuadrático.Estimación-Parámetros = t-Multivariable;

      Cuadrático.Estimación-Parámetros = ML
      CAMBIAR-A
      Cuadrático.Estimación-Parámetros = MVE;

      Cuadrático.Estimación-Parámetros = ML
      CAMBIAR-A
      Cuadrático.Estimación-Parámetros = Debiased;

      Cuadrático.Estimación-Parámetros = ML
      CAMBIAR-A
      Cuadrático.Estimación-Parámetros = Predictivo;

      Red-Neuronal.Weight.Decay = 0
      CAMBIAR-A
      Red-Neuronal.Weight.Decay = 0.001

      Red-Neuronal.Weight.Decay = 0
      CAMBIAR-A
      Red-Neuronal.Weight.Decay = 0.01
```

```
        Red-Neuronal.Weight.Decay = 0
        CAMBIAR-A
        Red-Neuronal.Weight.Decay = 0.1

        Vecinos-Próximos.Número-Vecinos = 1
        CAMBIAR-A
        Vecinos.Próximos.Número-Vecinos = 2

        Vecinos-Próximos.Número-Vecinos = 1
        CAMBIAR-A
        Vecinos.Próximos.Número-Vecinos = 3
END KNOWLEDGE-BASE Críticas-Modelos-Clasificadores;
```

Como se puede observar, en lo que se refiere a los discriminantes lineal y cuadrático, las reglas van encaminadas a probar estimadores de parámetros, alternativos al habitual ML (los mencionados en el apartado 4.1.3.1).

En las redes neuronales se fija el parámetro de *Weight decay* a los valores recomendados en [Ripley 94], entre $10^{-4} - 10^{-2}$. De esta manera, se reduce la complejidad de la red y con ello la varianza del clasificador.

Para los clasificadores de vecinos próximos, simplemente se cambia el número de vecinos a considerar, pasando de uno a dos o tres.

Inferencia: Modificar-Diseño

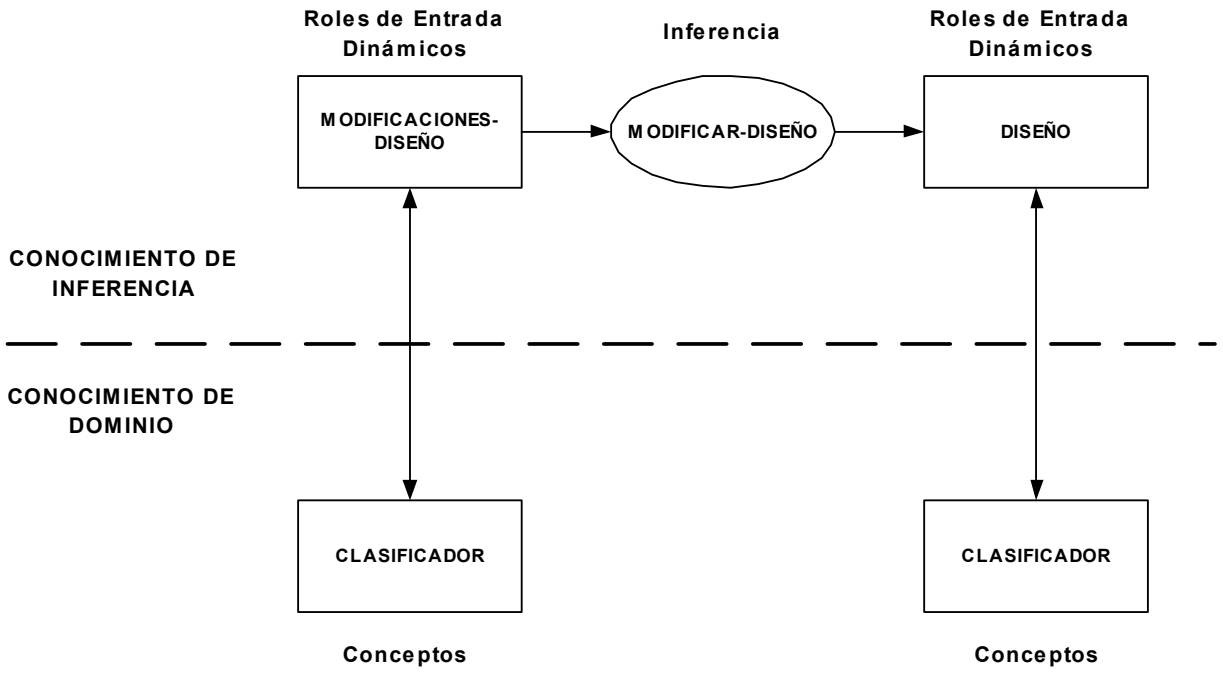

Figura 4.11. Inferencia Modificar-Diseño y su relación con el dominio

Objetivo: modificar el diseño, siguiendo los cambios sugeridos en Modificaciones-Diseño.

Se trata en definitiva de volver a entrenar y validar el modelo de clasificador, con los cambios introducidos.

Inferencia: Criticar-Restricciones

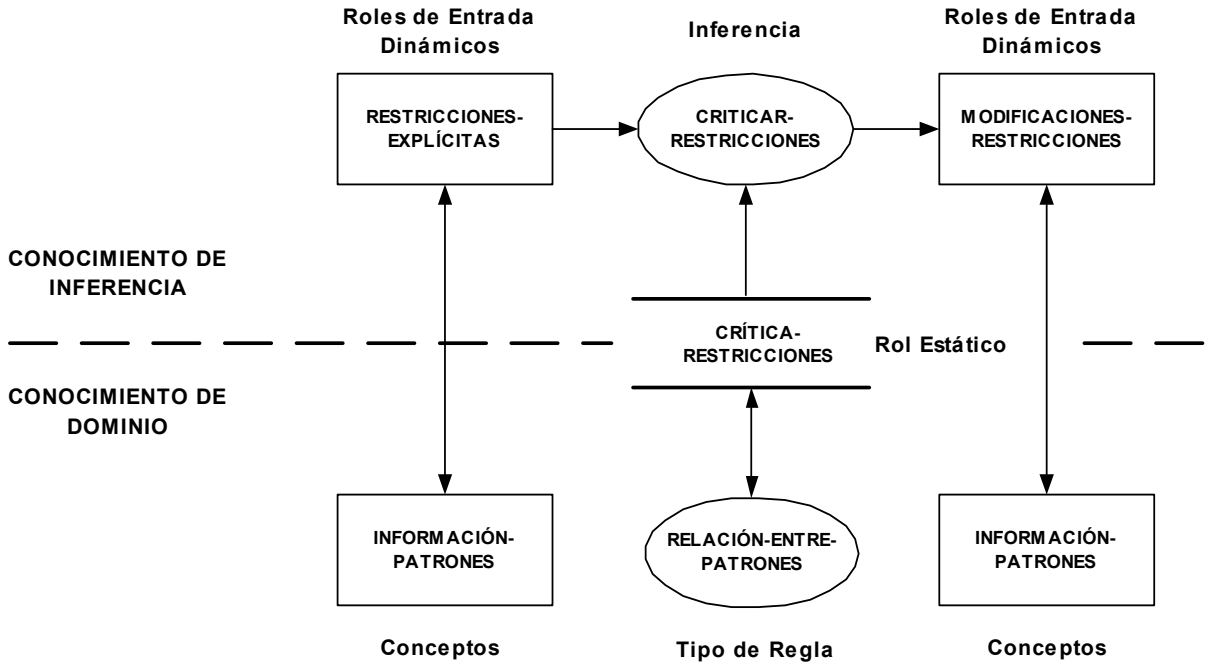

Figura 4.12. Inferencia Criticar-Restricciones y su relación con el dominio

Objetivo: criticar las restricciones, sugiriendo posibles modificaciones de las mismas.

La Base de Conocimiento asociada a esta inferencia es la siguiente:

```
KNOWLEDGE-BASE Críticas-Patrones-Entrada;
   USES: Relación-Entre-Patrones FROM Esquema-Dominio-Diseño-Clasificador;
   EXPRESSIONS:
      Información-Patrones.Reducción-Dimensionalidad = ninguno
      CAMBIAR-A
      Información-Patrones.Reducción-Dimensionalidad = Componentes-Principales;

      Información-Patrones.Reducción-Dimensionalidad = ninguno
      CAMBIAR-A
      Información-Patrones.Reducción-Dimensionalidad = Fisher;
END KNOWLEDGE-BASE Críticas-Patrones-Entrada;
```

Estas reglas cambian el atributo Reducción-Dimensionalidad, con el fin de sugerir diferentes métodos de reducción de la dimensionalidad de los patrones de entrada. Se

sugieren los métodos clásicos de componente principales y el basado en el discriminante de Fisher.

Inferencia: Modificar-Restricciones

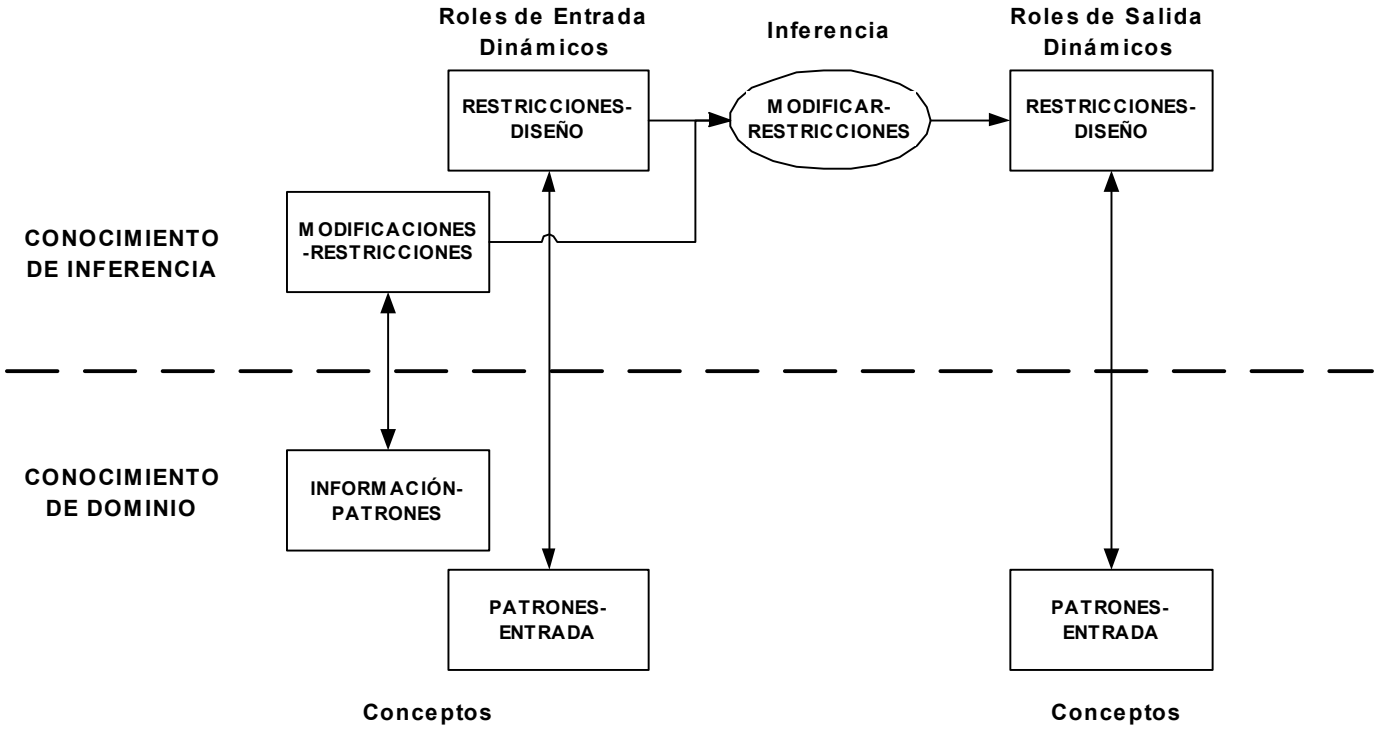

Figura 4.13. Inferencia Modificar-Restricciones y su relación con el dominio

Objetivo: modificar las restricciones, siguiendo las sugerencias en Modificaciones-Restricciones.

Como ya se explicó, supone un procesamiento de los patrones de entrada, orientado principalmente a una reducción de la dimensionalidad.

Inferencia: Ordenar-Diseños

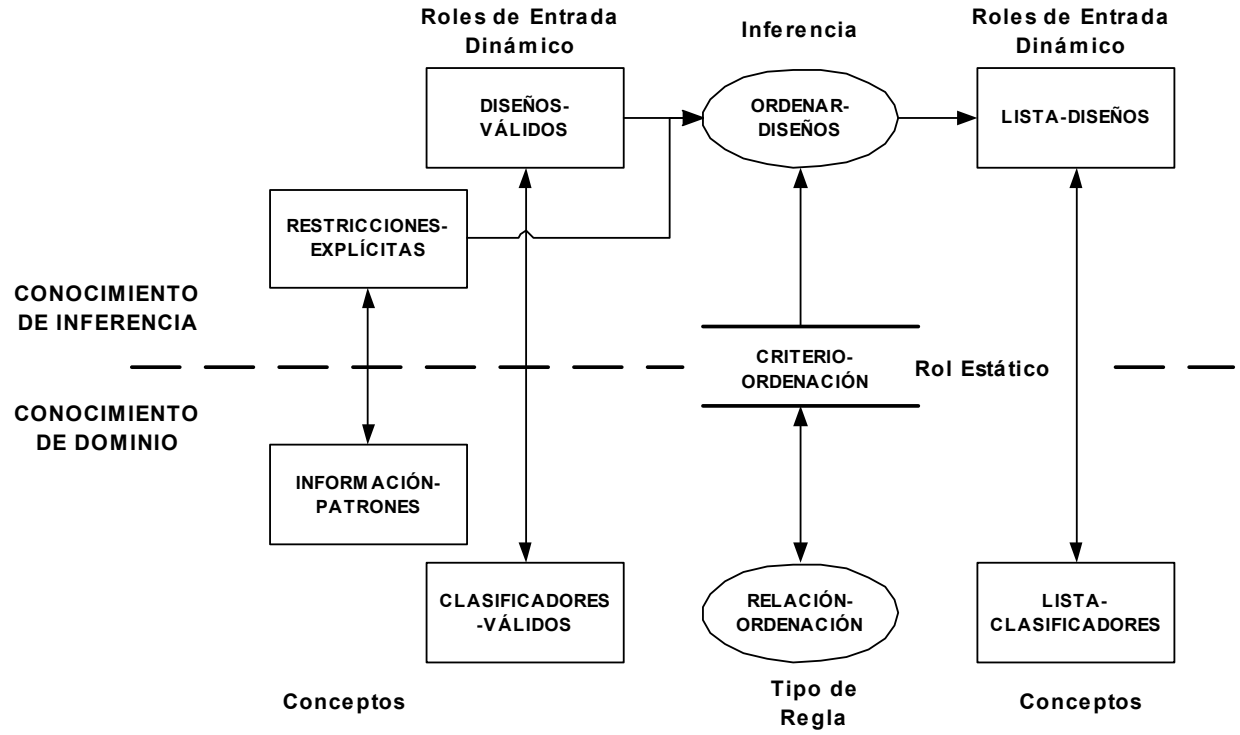

Figura 4.14. Inferencia ORDENAR-DISEÑOS y su relación con el dominio

Objetivo: ordenar los diseños válidos, siguiendo ciertos criterios.

Los criterios seguidos quedan reflejados en las reglas de la Base de Conocimiento Criterios-Ordenación-Clasificadores.

```
KNOWLEDGE-BASE Criterios-Ordenación-Clasificadores;
   USES: Relación-Ordenación FROM Esquema-Dominio-Diseño-Clasificador;
   EXPRESSIONS:
      Información-Patrones.Número-Patrones-Entrada >= 100 AND (Información-
Patrones.Test1-          Normalidad-Mono > 0.75 OR Información-Patrones.Test2-
Normalidad > 0.75) AND Información-                    Patrones.Test1-Normalidad-
Multi > 3.5
      ORDENAR-COMO
      Lineal.Índice-Ordenación = 1;

      Información-Patrones.Número-Clases >= 2
      ORDENAR-COMO
      Lineal.Índice-Ordenación = 1;
```

```
        Informacion-Patrones.Número-Patrones-Entrada >= 100 AND (Información-
Patrones.Test1-          Normalidad-Mono > 0.75 OR Información-Patrones.Test2-
Normalidad > 0.75) AND Información-                    Patrones.Test1-Normalidad-
Multi > 3.5
        ORDENAR-COMO
        Cuadrático.Índice-Ordenación = 2;

        Información-Patrones.Número-Clases >= 2
        ORDENAR-COMO
        Cuadrático.Índice-Ordenación = 2;

        Información-Patrones.Número-Clases >= 2
        ORDENAR-COMO
        Red-Neuronal.Índice-Ordenación = 3;

        Información-Patrones.Número-Clases >= 2
        ORDENAR-COMO
        Vecinos-Próximos.Índice-Ordenación = 4;
END KNOWLEDGE-BASE Criterios-Ordenación-Clasificadores;
```

En esta inferencia se ha optado por ordenar los diseños atendiendo a su "robustez", en el sentido de considerar primero los clasificadores más simples, esto es, con menos parámetros a determinar a partir de los datos.

Además, se ha añadido el criterio de normalidad de las distribuciones de las clases, formando parte del antecedente de las reglas que se refieren a los discriminantes lineal y cuadrático. La razón por la cual se ha utilizado ahora la condición de normalidad y no en la etapa de proponer, es porque creemos que no es un criterio que permita, a priori, descartar modelos de clasificadores, pero si puede resultar muy útil a la hora de ordenarlos.

De todas formas, parece evidente que el primer criterio a tener en cuenta a la hora de ordenar los clasificadores debería ser el error promedio calculado. Lo que ocurre es que, como ya se ha comentado, en la práctica, el cálculo de este error puede llegar a ser un proceso muy "ruidoso" y parece interesante disponer de otros criterios adicionales.

4.3.1.3 Método de Tarea

Hasta ahora, se ha descrito la parte estática del Modelo de Conocimiento. A continuación se muestra el Método de Tarea para el problema de diseño, donde, se recoge la dinámica de las diferentes inferencias y la estrategia seguida.

TASK-METHOD Diseño-Usando-Proponer-Criticar-Modificar;
REALIZES: Diseño-Clasificador;
DECOMPOSITION:
INFERENCES: Explicitar-Restricciones, Especificar-Estructura-Diseño, Proponer-Diseño, Verificar-Diseño, Criticar-Diseño, Modificar-Diseño, Criticar-Restricciones, Modificar-Restricciones, Ordenar-Diseños;
ROLES:
INTERMEDIATE: Restricciones-Explícitas, Estructura-Diseño, Diseño, Valor-Verdad, Prestaciones-Diseño, Violaciones-Diseño, Modificaciones-Diseño, Modificaciones-Restricciones, Diseños-Válidos, Lista-Diseños;
CONTROL-STRUCTURE:
Explicitar-Restricciones(Restricciones-Diseño → Restricciones-Explícitas);
WHILE NEW-SOLUTION Especificar-Estructura-Diseño(Restricciones-Explícitas → Estructura-Diseño) **DO**
Proponer-Diseño(Estructura-Diseño + Restricciones-Diseño → Diseño);
Verificar-Diseño(Diseño + Prestaciones-Diseño → Violaciones-Diseño + Diseños-Válidos + Valor-Verdad);
WHILE Valor-Verdad == **FALSE AND NEW-SOLUTION** Criticar-Diseño(Diseño+Violaciones-Diseño → Modificaciones-Diseño) **DO**
Modificar-Diseño(Modificaciones-Diseño → Diseño);
Verificar-Diseño(Diseño + Prestaciones-Diseño → Violaciones-Diseño + Diseños-Válidos + Valor-Verdad);
END-WHILE
END WHILE
IF EMPTY Diseños-Válidos **THEN**
WHILE NEW-SOLUTION Criticar-Restricciones(Restricciones-Explícitas → Modificaciones-Restricciones) **DO**
Modificar-Restricciones(Restricciones-Diseño + Modificaciones-Restricciones → Restricciones-Diseño);
Explicitar-Restricciones(Restricciones-Diseño → Restricciones-Explícitas);
WHILE NEW-SOLUTION Especificar-Estructura-Diseño(Restricciones-Explícitas → Estructura-Diseño) **DO**
Proponer-Diseño(Estructura-Diseño + Restricciones-Diseño → Diseño);
Verificar-Diseño(Diseño + Prestaciones-Diseño → Violaciones-Diseño + Diseños-Válidos + Valor-Verdad);
WHILE Valor-Verdad == **FALSE AND NEW-SOLUTION** Criticar-Diseño(Diseño+Violaciones-Diseño → Modificaciones-Diseño) **DO**
Modificar-Diseño(Modificaciones-Diseño → Diseño);

```
        Verificar-Diseño(Diseño + Prestaciones-Diseño → Violaciones-Diseño + Diseños-
Válidos +           Valor-Verdad);
        END-WHILE
        END WHILE
        END WHILE
        END IF
        Ordenar-Diseños(Diseños-Validos → Lista-Diseños);
END TASK-METHOD Diseño-Usando-Proponer-Criticar-Modificar;
```

La estrategia seguida consiste entonces en los siguientes pasos:

a) Proponer diseño, si quedan propuestas, si no, ir al paso d).
b) Verificarlo. Si el diseño es válido, guardarlo, si no, criticar el diseño, sugiriendo posibles modificaciones del mismo.
c) Modificar el diseño, probando las diferentes sugerencias y verificarlo cada vez.
d) Si no hay diseños válidos, criticar restricciones, sugiriendo posibles modificaciones de las mismas, si no, ir al paso f).
e) Mientras queden sugerencias, repetir los pasos de a) a c), si no, ir al paso f).
f) Ordenar los diseños válidos.

4.3.2 Implementación del Modelo de Conocimiento en el Problema de Diseño de un Clasificador. Modelo de Diseño del CommonKADS

En el capítulo 2, se explicó con detalle el Modelo de Diseño de la metodología CommonKADS. Dentro de este modelo, se proponía una arquitectura software fundamentada en el lenguaje CLIPS, que soportaba la implementación del Modelo de Conocimiento, generando una serie de ficheros que facilitaban enormemente esta labor. Los ficheros específicos para este problema figuran en el apéndice.

La novedad más importante respecto a lo ya expuesto, viene dada por la necesidad de incorporar algún lenguaje adecuado para codificar los algoritmos (en ocasiones, bastante complejos) utilizados en algunas de las inferencias. El lenguaje elegido ha sido el R, que pasamos a describir a continuación, así como la forma en la que se ha integrado con el CLIPS.

4.3.2.1 El Lenguaje R

R [R 00] es un lenguaje que proporciona grandes facilidades para el procesamiento y la visualización de datos.

Algunas de sus características más notables son:

- Manipulación y almacenamiento de datos eficiente.
- Extensa colección de operadores preparados para trabajar con matrices.
- Gran cantidad de herramientas para análisis de datos.
- Funciones gráficas diversas.
- Lenguaje de programación con facilidades tales como sentencias de control, entrada-salida,...

R constituye una reimplementación del lenguaje S [Venables 99], desarrollado por AT&T por Rick Becker, John Chambers y Allan Wilks, pero a diferencia de éste, se trata de software de libre distribución.

Una de las grandes ventajas del R es la existencia de gran cantidad de paquetes adicionales (packages) a la configuración básica. Estos paquetes extienden el lenguaje y le dan una potencialidad muy grande, en particular, en el área del reconocimiento de patrones.

Estos motivos, principalmente, son los que nos han llevado a elegir este lenguaje en la implementación de los algoritmos utilizados en las inferencias: Explicitar-Restricciones, Proponer-Diseño, Modificar-Diseño y Modificar-Restricciones.

4.3.2.2 Interface entre el R y el CLIPS

A la vista de lo comentado en el apartado anterior, en la ejecución de ciertas inferencias será preciso llamar a funciones del lenguaje R que implementen los algoritmos necesarios. Para hacer compatible esta circunstancia con el soporte en CLIPS desarrollado de forma genérica, se ha establecido una interface entre ambos lenguajes. Esta interface permite hacer llamadas a funciones R desde código CLIPS, de forma que

los resultados que devuelvan puedan ser almacenados en los objetos CLIPS correspondientes, recuperando además el control de la ejecución.

En Linux es posible compilar el R como una librería independiente para ser enlazada con otras aplicaciones. Esto permite, utilizando ciertas funciones especiales, poder ejecutar código R, desde dentro de otra aplicación. Por este motivo, se eligió el sistema operativo Linux como entorno de trabajo. Por otro lado, también es posible compilar el CLIPS como una librería independente y ser llamado desde dentro de otra aplicación.

Teniendo en cuenta ambas posibilidades, se procedió a compilar en C, tanto el R como el CLIPS, y luego se enlazaron con un pequeño programa principal de inicialización. De esta manera se consiguió el objetivo deseado de integrar ambos lenguajes. En la figura 4.15 se esquematiza el interface implementado.

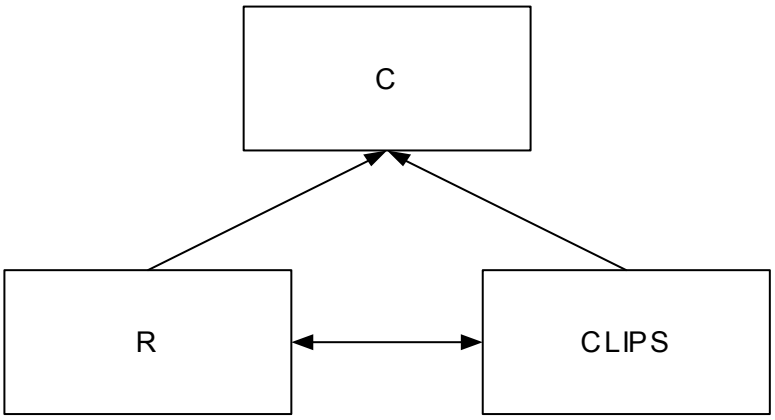

Figura 4.15. Esquema de interface entre el R y el CLIPS

4.4 Validación del SBC para Diseño de Clasificadores

Nuestro sistema para el diseño de clasificadores va a ser validado utilizando diferentes conjuntos de datos. En las secciones siguientes se comentan los resultados encontrados en cada caso. De entre los conjuntos de datos seleccionados, destacan dos en particular.

Por un lado, se dispone de datos de registros de Electroencefalogramas (EEG) correspondientes a diferentes pacientes y que han sido capturados en el hospital Nuestra Señora de la Candelaria en Tenerife, concretamente en el Departamento de Neurofisiología. Algunos de estos pacientes padecen la enfermedad de Alzheimer y se pretende entonces, diseñar un clasificador que permita distinguir diagnosticar esta patología. Realmente, la automatización del diagnóstico de patologías cerebrales

constituye una línea de investigación importante del Grupo de Computadoras y Control de la Universidad de La Laguna. Esta línea se inició a principios de los 90 y ha dado lugar a numerosos trabajos de investigación [Sánchez 93] [Mañas 94] [Piñeiro 96].

Por otro lado, se tuvo acceso a datos de niños con dislexia, recogidos por miembros del Departamento de Psicología Evolutiva y de la Educación de la Universidad de La Laguna. De nuevo, el Sistema Basado en el Conocimiento, propondrá los clasificadores que considere mejores para detectar este problema.

4.4.1 Aplicación a Datos de EEG de Pacientes con Alzheimer

Antes de entrar a comentar los resultados obtenidos con estos datos, será conveniente introducir algunos conceptos de Neurofisiología.

4.4.1.1 Origen de la Actividad Eléctrica Cerebral

El cerebro humano está compuesto por la agregación de un número elevado de neuronas con conectividades muy altas, lo que produce un número astronómico de conexiones. Los fenómenos en los que está basado el proceso de la información en la neurona son de naturaleza electro-química. El registro de esa actividad colectiva eléctrica cerebral constituye el electroencefalograma (EEG). Dicho registro se realiza fijando electrodos sobre el cuero cabelludo y amplificando las variaciones de potencial (de muy pequeña magnitud, del orden de decenas de μV) en el llamado electroencefalógrafo, que es el sistema encargado además de registrar las señales. Las variaciones de potencial son el resultado de la actividad conjunta de las células que pertenecen al córtex cerebral, la capa más externa del cerebro. Aunque el grosor del córtex es de sólo 2 mm., posee una superficie de unos 2000 cm^2, debido a que presenta una gran cantidad de circunvoluciones que desarrollan esa gran superficie en un pequeño volumen. En el córtex es donde residen las funciones superiores del cerebro y uno de los órganos que ha experimentado un mayor desarrollo evolutivo respecto a nuestros ancestros.

La actividad eléctrica cerebral fue descubierta en el hombre en 1924, por el psiquiatra Hans Berger, quien describió por primera vez los ritmos cerebrales en 1929.

Las señales cerebrales presentan aspectos distintos según el estado del sujeto: en reposo, realizando tareas mentales, durante el sueño, etc. Además se manifiestan de

acuerdo siguiendo una topología determinada, es decir, varían según el punto concreto de la cabeza donde se registren.

El conocimiento sobre el funcionamiento de los mecanismos cerebrales que originan estas señales es muy incompleto. Por ejemplo, se conoce que hay determinadas áreas subcorticales (núcleos talámicos) que provocan la actividad sincronizada de un gran número de neuronas del córtex, constituyendo una especie de marcapasos. Son las actividades simultáneas y sincronizadas en gran cantidad de neuronas individuales las que dan lugar al EEG macroscópico.

A pesar del escaso conocimiento sobre su origen, el EEG constituye una herramienta de primer orden en el diagnóstico de patologías cerebrales. A diferencia de los métodos de visualización de las estructuras cerebrales, como la tomografía axial computerizada, angiografía, etc., que poseen resoluciones espaciales superiores a la del EEG, éste proporciona información funcional del cerebro en una prueba relativamente rápida y no invasiva. De esta manera se pueden detectar patologías que no se reflejan en cambios estructurales sino en alteraciones de la función cerebral que sí se registran en el EEG.

4.4.1.2 Descripción de las Señales EEG

Los elementos observados en el EEG se suelen estudiar describiendo las señales en términos de *ondas*, cuyo significado en neurofisiología es simplemente el segmento de la señal entre dos máximos o dos mínimos. Las características fundamentales de las ondas son su duración, medida en ms., y la amplitud (distancia entre el máximo y mínimo de la onda). Las ondas y series de ondas se suelen clasificar en bandas de frecuencia. Una serie de ondas que aparecen un cierto tiempo con frecuencia, morfología y localización determinadas se suele denominar *ritmo*. Un grupo de ondas que aparece y desaparece abruptamente, diferenciándose claramente en su frecuencia, amplitud y morfología de la actividad de base se denomina *paroxismo* y no necesariamente es una actividad patológica. La figura 4.16 presenta un registro de 16 canales de unos 10 s. de duración, con actividad normal EEG, recogido con el sujeto en reposo con los ojos cerrados. El registro ha sido tomado con el montaje bipolar que aparece en la figura 4.17, que es un montaje usado en la rutina clínica. En un estado relajado, como en el que ha sido tomado el registro, la mayoría de la población presenta

actividad del ritmo alfa, que son los husos sinusoidales que predominan en la figura. Se aprecia una diferenciación topológica, es decir, una variación de la actividad según el canal y por tanto la zona de la cabeza donde se ha recogido la señal. Por ejemplo, la actividad alfa se hace más patente en los canales 14 y 16, que representan la señal en la parte posterior de la cabeza (electrodos occipitales respecto al vértex C_z). Es evidente también la variación en el tiempo de los ritmos observados, que van cambiando de amplitud.

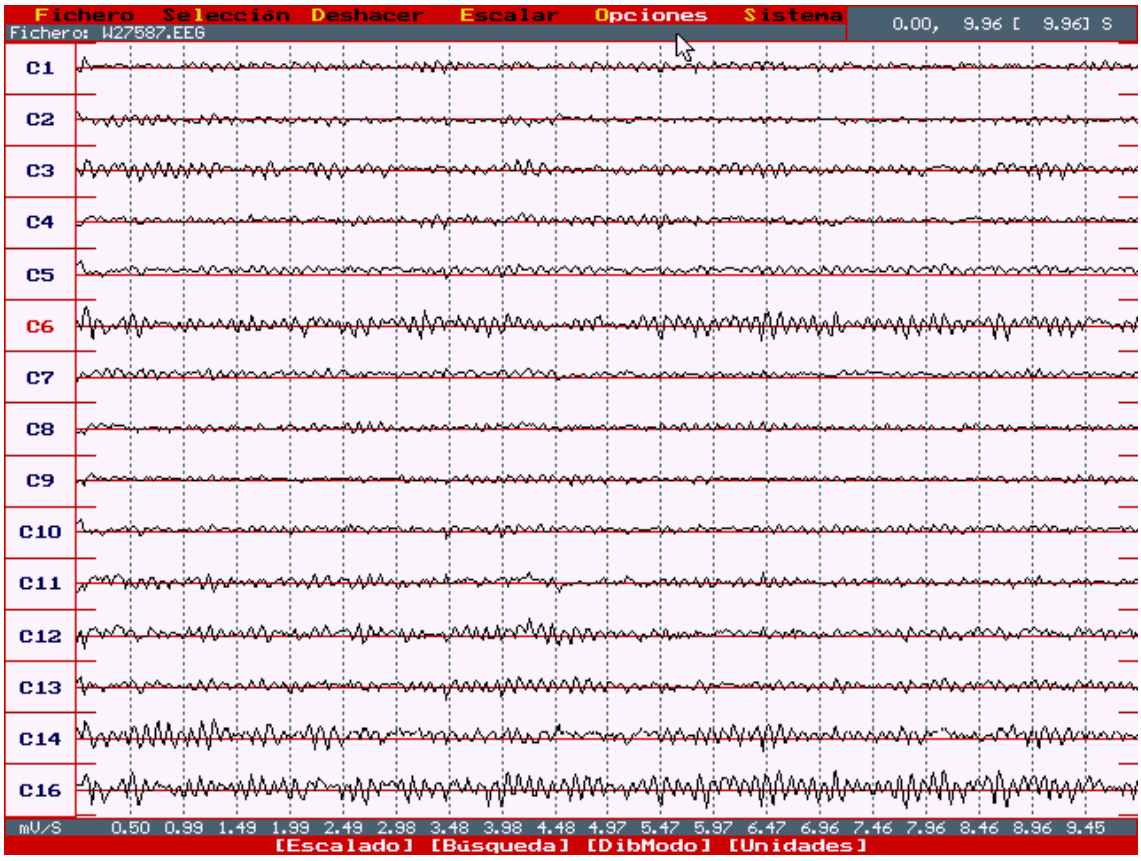

Figura 4.16. Registro EEG de unos 10 s. de duración tomado sobre 16 canales.

Además de estas señales, se pueden apreciar en los registros EEG otras actividades eléctricas cuyo origen no está en el cerebro y que perturban el diagnóstico (artefactos). Como ejemplos se pueden citar la actividad debida al parpadeo, al movimiento de la mandíbula, la inducida por perturbaciones electromagnéticas externas, etc. El especialista sabe identificar estos artefactos, así como los distintos transitorios en su examen del EEG.

Hay que destacar además que, aún tratándose de los ritmos más estables, las señales cerebrales frecuentemente no son estacionarias. Por ejemplo, la aparición de un

ritmo de una frecuencia determinada puede implicar la desaparición (bloqueo) de otro, activo hasta entonces, o se pueden producir cambios más lentos en los espectros a lo largo del registro.

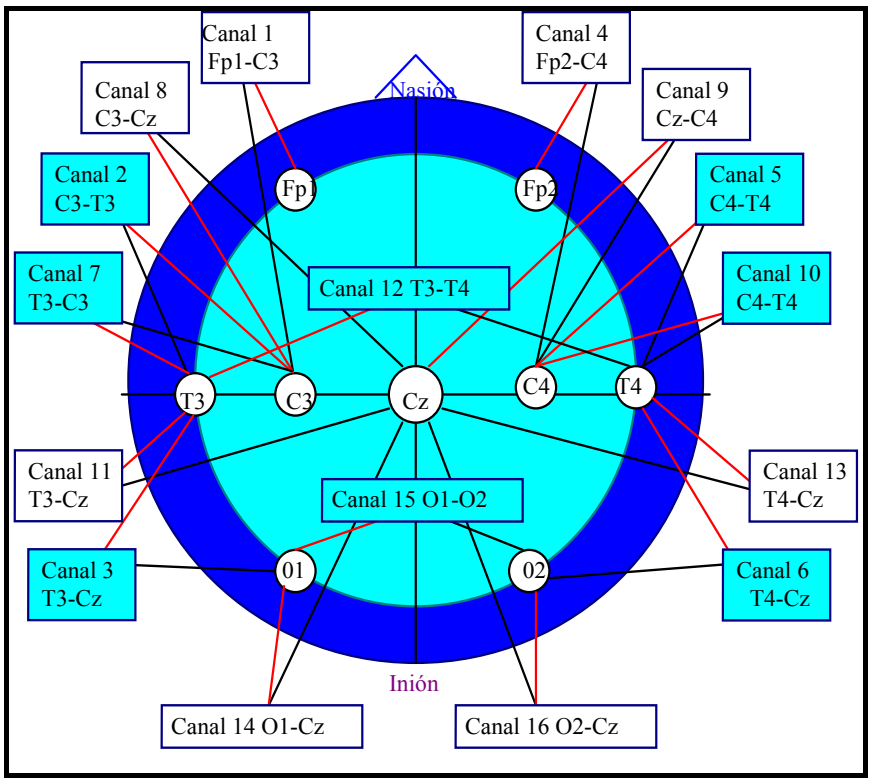

Figura 4.17. Montaje utilizado en los registros EEG y PEV.

Se suele distinguir entre los ritmos de fondo, que aparecen en bandas de frecuencia, duración, morfología, y posiciones bien definidas frente a otras actividades transitorias. Los ritmos de fondo aparecen tradicionalmente asociados a las bandas de frecuencia de su período fundamental: Alfa, de 8 a 13 Hz, Beta, actividad mayor de 13 Hz apreciable en las regiones frontal y central, Theta, de frecuencias entre 4 y 7 Hz suele aparecer en áreas central, parietal y temporal. Estos ritmos se pueden apreciar en EEGs de vigilia, ya que el sueño tiene su propia clasificación de actividades.

4.4.1.3 Demencias

La demencia es un síndrome que se caracteriza por la pérdida adquirida de las habilidades cognoscitivas y emocionales de tal intensidad que interfiere con el funcionamiento diario y con la calidad de vida.

El término demencia no indica alguna causa específica ya que más de 55 entidades clínicas pueden ocasionarla, algunas de ellas de carácter no progresivo. La prevalencia de la demencia es del 1 % en la población general a la edad de 60 años y se duplica cada 5 años hasta alcanzar del 30% a 50% a los 85 años de edad. Cuando confrontamos a un paciente con este síndrome, es esencial definir el tipo de demencia para determinar el tratamiento e informar acerca del pronóstico, los posibles riesgos genéticos y planear el cuidado del paciente.

La tomografía axial computerizada (TAC) de cráneo y la imagen de resonancia magnética (IRM) del encéfalo son utilizadas para excluir lesiones estructurales como causa de demencia tales como infarto cerebral, tumores, hidrocefalia, hematomas extracerebrales etc. Se considera que la TAC debe solicitarse cuando el paciente con deterioro cognoscitivo no presenta anormalidades en el examen neurológico; en cambio, si existe disfunción motora asociada, rigidez o asimetría en los reflejos, se deberá solicitar IRM. Al igual que con los exámenes de laboratorio, con los estudios de neuroimagen es bajo el porcentaje de diagnóstico de demencias tratables, aunque el beneficio de diagnosticar algún paciente con lesión estructural no detectada clínicamente compensa la frecuente ausencia de anormalidades en la neuroimagen.

El electroencefalograma (EEG) en la evaluación de la demencia no se utiliza en forma rutinaria pero puede ayudar en la identificación de trastornos metabólicos o tóxicos, crisis parciales complejas, crisis epilépticas sin manifestación convulsiva o enfermedad de Creutzfeldt-Jakob (demencia rápidamente progresiva). El líquido cefalorraquídeo (LCR) debe analizarse en casos atípicos (demencia en pacientes jóvenes con curso subagudo o con signos de enfermedad sistémica). Existen marcadores biológicos para la enfermedad de Alzheimer que pueden investigarse en el LCR como son la proteína tau y el beta-amiloide aun cuando su valor diagnóstico en la actualidad es incierto y no se recomienda para uso rutinario. La prueba para el virus de la inmuodeficiencia humana (VIH) se sugiere efectuar en pacientes con factores de riesgo conocidos para SIDA, puesto que la demencia se presenta hasta en 20% de estos pacientes.

Enfermedad de Alzheimer

Representa el 70% de los casos de demencia, tiene un curso progresivo con supervivencia de 8 a 10 años. El rasgo cognoscitivo característico es el deterioro progresivo de la memoria para hechos recientes. Hay desorientación de carácter progresivo en tiempo y en lugar. El deterioro del lenguaje, signo importante de esta enfermedad, se inicia con dificultad para encontrar palabras durante la comunicación espontánea, progresando hasta un lenguaje vago utilizando frases automáticas. Se pierde la habilidad para nombrar objetos (anomia), se deteriora la comprensión verbal y la habilidad para efectuar actividades de la vida diaria (conducir, uso de utensilios etc.) tanto por la apraxia y defectos en atención visual, como por la pérdida de la relación espacial y detección del movimiento. Posteriormente, aparece acalculia e incapacidad de efectuar actos complejos útiles en la higiene personal y en la preparación de alimentos.

Los síntomas no-cognoscitivos o de conducta son importantes porque producen más problemas a quienes se encargan del cuidado del paciente. Estos fluctúan desde pasividad hasta agresividad muy marcada. Existe disminución en la expresión emocional, pérdida de la iniciativa e incremento de la suspicacia. Las ilusiones paranoides (acusaciones de robo, infidelidad marital y persecución) afectan hasta el 50% de los pacientes con enfermedad de Alzheimer. Las alucinaciones usualmente de tipo visual (ver parientes ya fallecidos, intrusos, animales), ocurren en 25% de los casos. La depresión y ansiedad se observan en el 40% de los casos y pueden ser el signo inicial de la enfermedad. En etapas iniciales el examen neurológico es normal. El inicio temprano de signos extrapiramidales (rigidez y temblor) indica la presencia de una forma atípica de demencia conocida como variante con cuerpos de Lewy.

4.4.1.4 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

El número de datos de los que se dispuso no era demasiado grande: 36 pacientes de control y 42 pacientes con Alzheimer. Como características se eligieron los valores de potencia espectral relativa en ciertos instantes de tiempo característicos, en los que esta potencia se hacía máxima en las cuatro bandas de frecuencia consideradas:

	Límite Inferior (Hz)	Límite Superior (Hz)
Delta	1.5	3.5
Theta	3.5	7.5
Alfa1	7.5	9.5
Alfa2	9.5	12.5

Tabla 4.1. Bandas de frecuencia consideradas

Esto nos da un conjunto de 16 parámetros por canal. Como se han tomado dos canales: el 12 y el 15, por considerarlos más significativos en este problema (de acuerdo con el criterio del médico), nos quedamos con 32 características.

El concepto Información-Patrones para estos datos quedaría de la siguiente manera:

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {36, 42};     /*en cada clase*/
      Número-Clases: 2;    /*nº de clases*/
      Número-Características: 32;            /*nº de características*/
      Test1-Normalidad-Mono: {0.0024, 0.0000};          /*probabilidades posteriori en
cada clase*/
      Test2-Normalidad-Mono: {0.0000, 0.0000};          /*probabilidades posteriori en
cada clase*/
      Test1-Normalidad-Multi: {0.0000, 0.0000};      /*valores P en cada clase*/
      Distancia-Bhattacharyya: 12.0401;      /*para dos clases*/
      Distancia-Bhattacharyya-1: 2.1149;     /*para dos clases*/
      Distancia-Bhattacharyya-2: 9.9253;     /*para dos clases*/
      Sesgo-Lineal: 0.0000;/*para dos clases*/
      Sesgo-Cuadrático: 0.0000;        /*para dos clases*/
      Dimensionalidad-Intrínseca: 17;
      Reducción-Dimensionalidad: ninguno   /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

Los tests de normalidad expresan claramente que los datos no siguen una distribución normal. Además la distancia de Bhattacharyya nos indica que existe una gran separabilidad entre clases, tanto en medias como en covarianzas. Es este factor el que

compensa la excesiva dimensionalidad, para dar unos términos de sesgo lineal y cuadrático muy pequeños.

En estas condiciones, la estrategia seguida por el SBC prioriza el diseño basado en los discriminantes lineal y cuadrático, con los que se consigue un error promedio de aproximadamente un 1% (con probabilidades a priori iguales), esto es, un excelente resultado como sistema de diagnóstico del Alzheimer.

4.4.2 Aplicación a Datos de Niños con Dislexias

Antes de entrar a comentar los resultados obtenidos con estos datos, será conveniente introducir algunos conceptos relacionados con la Dislexia.

4.4.2.1 Introducción al Problema de la Dislexia

Etimológicamente la palabra dislexia quiere decir aproximadamente dificultades de lenguaje. En la acepción actual se refiere a problemas de lectura o trastorno en la adquisición de la lectura.

Una primera definición sencilla de la dislexia es la que nos dice que es el problema para aprender a leer que presentan niños cuyo coeficiente intelectual es normal y no aparecen otros problemas físicos o psicológicos que puedan explicar dichas dificultades.

Según algunas estadísticas, la dislexia afecta en mayor o menor grado a un 10% o un 15% de la población escolar y adulta. Hay consenso en que entre un 4 y un 5% de los niños presentan problemas graves de aprendizaje de la lectura, con la consecuente dificultad a la hora de escribir.

Dada la generalización de la enseñanza a toda la población de forma obligatoria y el uso prioritario de la lectura y la escritura como mediadores de la enseñanza, la cantidad de niños que tienen dificultades escolares por esta causa es un factor relevante a tener en cuenta por el personal docente. Según las estadísticas citadas arriba se puede esperar que en cada aula de 25 alumnos haya al menos un niño con esta dificultad para el aprendizaje.

Según M.Thomson, la Dislexia es una grave dificultad con la forma escrita del lenguaje, que es independiente de cualquier causa intelectual, cultural y emocional. Se caracteriza porque las adquisiciones del individuo en el ámbito de la lectura, la escritura y el deletreo, están muy por debajo del nivel esperado en función de su inteligencia y de su edad cronológica. Es un problema de índole cognitivo, que afecta a aquellas habilidades lingüísticas asociadas con la modalidad escrita, particularmente el paso de la modalidad escrita, particularmente el paso de la codificación visual a la verbal, la memoria a corto plazo, la percepción de orden y la secuenciación.

Existe cierta confusión en el uso de "apellidos " aplicados a la dislexia. Tales los calificativos de madurativa, evolutiva o adquirida.

En la práctica se habla de dislexia evolutiva cuando aparecen dificultades y síntomas parecidos o iguales a los disléxicos, en niños que inician su aprendizaje. Lo que ocurre es que rápidamente estos síntomas desaparecen por sí solos. Estos síntomas pueden ser inversiones en la escritura y/o en la lectura, adiciones, omisiones, escritura en espejo, vacilaciones, repeticiones...

El calificativo de dislexia madurativa se da a las dificultades de aprendizaje de la lecto-escritura que se dan en niños con deficiencias intelectuales.

La dislexia va unida en ocasiones a otros problemas de aprendizaje escolar, tales como la disgrafía (dificultades en el trazado correcto de las letras, en el paralelismo de las líneas, en el tamaño de las letras, en la presión de la escritura...) y en fases posteriores aparece la disortografía (dificultades para el uso correcto de las reglas de ortografía, desde las que se llaman de ortografía natural a las de nivel más complejo.)

En ocasiones la dislexia va unida a dificultades de pronunciación, con mayor incidencia en la dificultad de pronunciación de palabras nuevas, largas o que contengan combinaciones de letras del tipo de las que le producen dificultades en la lectura.

La dislexia se presenta en muchos grados, desde pequeños problemas superables en breve plazo, hasta una dificultad que se arrastra de por vida y que se aproxima como en un continuo hacia la disfasia, que es un problema más grave y profundo de todas las áreas de lenguaje. De cualquier modo, con la iniciación del tratamiento con suficiente precocidad se suelen derivar resultados positivos y una clara mejora en el rendimiento escolar. La mayor o menor efectividad va a depender de factores tales como la profundidad del trastorno, el nivel de motivación inicial o el que se le consiga inculcar,

grado de implicación de la familia y el profesorado, adecuado diagnóstico y tratamiento, duración y seguimiento del trabajo...

Profundizando en la detección de los niños con problemas de dislexia, de acuerdo con los criterios de la Asociación Británica de Dislexia y con otras fuentes, los signos que pueden tener (algunos de ellos, no necesariamente todos) los niños según la edad serían los siguientes:

Niños de Preescolar (Educación Infantil)

- Historia Familiar de problemas disléxicos (padres, hermanos, otros familiares).
- Retraso en aprender a hablar con claridad.
- Confusiones en la pronunciación de palabras que se asemejan por su fonética.
- Falta de habilidad para recordar el nombre de series de cosas, por ejemplo los colores.
- Confusión en el vocabulario que tiene que ver con la orientación espacial
- Alternancia de días "buenos" y "malos " en el trabajo escolar, sin razón aparente.
- Aptitud para la construcción y los objetos y juguetes "técnicos" (mayor habilidad manual que lingüística, que aparecerá típicamente en las pruebas de inteligencia.), juegos de bloques, lego.
- Dificultad para aprender las rimas típicas del preescolar.
- Dificultades con la palabras rimadas.
- Dificultades con las secuencias.

  Niños hasta 9 años
- Particular dificultad para aprender a leer y escribir.
- Persistente tendencia a escribir los números en espejo o en dirección o orientación inadecuada.
- Dificultad para distinguir la izquierda de la derecha.
- Dificultad de aprender el alfabeto y las tablas de multiplicar y en general para retener secuencias, como por ejemplo los días de la semana, los dedos de la mano, los meses del año.
- Falta de atención y de concentración.
- Frustración, posible inicio de problemas de conducta.

Niños entre 9 y 12 años

- Continuos errores en lectura, lagunas en comprensión lectora.
- Forma extraña de escribir, por ejemplo, con omisiones de letras o alteraciones del orden de las mismas.
- Desorganización en casa y en la escuela.
- Dificultad para copiar cuidadosamente en la pizarra y en el cuaderno.
- Dificultad para seguir instrucciones orales.
- Aumento de la falta de autoconfianza y aumento de la frustración.
- Problemas de comprensión del lenguaje oral e impreso.
- Problemas conductuales: impulsividad, corto margen de atención , inmadurez.

Niños de 12 años en adelante.

- Tendencia a la escritura descuidada, desordenada, en ocasiones incomprensible.
- Inconsistencias gramaticales y errores ortográficos, a veces permanencia de las omisiones, alteraciones y adiciones de la etapa anterior.
- Dificultad para planificar y para redactar relatos y composiciones escritas en general.
- Tendencia a confundir las instrucciones verbales y los números de teléfono.
- Gran dificultad para el aprendizaje de lenguas extranjeras.
- Baja autoestima.
- Dificultad en la percepción del lenguaje, por ejemplo en seguir instrucciones.
- Baja comprensión lectora.
- Aparición de conductas disruptivas o de inhibición progresiva. A veces, depresión.
- Aversión a la lectura y la escritura.

4.4.2.2 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

En este caso la población de estudio está compuesta por 101 sujetos con dislexia y 43 de control. Como características discriminatorias se eligieron los tiempos de reacción a palabras y pseudopalabras (palabras que no significan nada pero son fonológicamente correctas), contenidas en diferentes tests de destreza estándar TALE (Test de Análisis de Lectoescritura) [Toro 80].

El concepto Información-Patrones para estos datos quedaría de la siguiente manera:

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {43, 101};   /*en cada clase*/
      Número-Clases: 2;    /*nº de clases*/
      Número-Características: 2;              /*nº de características*/
      Test1-Normalidad-Mono: {0.0340, 0.0000};              /*probabilidades posteriori en
cada clase*/
      Test2-Normalidad-Mono: {0.0301, 0.0001};              /*probabilidades posteriori en
cada clase*/
      Test1-Normalidad-Multi: {0.0001, 0.0000};      /*valores P en cada clase*/
      Distancia-Bhattacharyya: 0.6570;       /*para dos clases*/
      Distancia-Bhattacharyya-1: 0.1531;     /*para dos clases*/
      Distancia-Bhattacharyya-2: 0.5069;     /*para dos clases*/
      Sesgo-Lineal: 0.0000;/*para dos clases*/
      Sesgo-Cuadrático: 0.0000;       /*para dos clases*/
      Dimensionalidad-Intrínseca: 2;
      Reducción-Dimensionalidad: ninguno   /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

Al igual que con los datos del EEG, los tests de normalidad expresan claramente que los datos no siguen una distribución normal. Atendiendo a la distancia de Bhattacharyya, no parece que las clases estén excesivamente separadas, si bien, en este caso la dimensionalidad es muy pequeña y no genera apenas sesgo en los discriminantes lineal y cuadrático.

En estas condiciones, la estrategia seguida por el SBC no propondría el discriminante lineal pero si el cuadrático (la separabilidad viene más bien de las covarianzas de las clases). En cualquier caso, los errores más bajos se consiguieron, por un lado, con la red neuronal con $3n$ neuronas (seis en este caso) en la capa oculta y parámetro de *Weight-decay* igual a 10^{-2}, y por otro lado, con los clasificadores de vecinos próximos con dos y tres vecinos. Estamos hablando de un error de aproximadamente un 25%.

4.4.3 Aplicación a los Datos Iris de Fisher

Se trata del conjunto de datos más famoso, y de alguna manera estándar, en el campo de la probabilidad y estadística. Fueron recogidos por Anderson, y posteriormente utilizados por Fisher [Fisher 36].

Los datos se corresponden a medidas en centímetros de la longitud y anchura del sépalo y la longitud y anchura del pétalo de tres especies de flores Iris: Iris Setosa, Iris Versicolor e Iris Virginica (50 datos en cada clase).

4.4.3.1 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {50, 50, 50};/*en cada clase*/
      Número-Clases: 3;    /*nº de clases*/
      Número-Características: 4;              /*nº de características*/
      Test1-Normalidad-Mono: {0.0997, 0.3722, 0.3776};              /*probabilidades
posteriori en cada clase*/
      Test2-Normalidad-Mono: {0.1285, 0.5095, 0.4946};              /*probabilidades
posteriori en cada clase*/
      Test1-Normalidad-Multi: {0.0026, 0.6357, 0.4952};     /*valores P en cada clase*/
      Distancia-Bhattacharyya: 1.9643;      /*para dos clases*/
      Distancia-Bhattacharyya-1: 1.7774;    /*para dos clases*/
      Distancia-Bhattacharyya-2: 0.1870;    /*para dos clases*/
      Sesgo-Lineal: 0.0947;/*para dos clases*/
      Sesgo-Cuadrático: 0.2590;       /*para dos clases*/
      Dimensionalidad-Intrínseca: 3;
      Reducción-Dimensionalidad: ninguno   /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

En este caso, es bien conocido que la separabilidad de las medias es bastante grande (sobre todo entre dos de las clases), y por eso con un discriminante lineal se puede conseguir un excelente resultado, concretamente una probabilidad promedio de error de aproximadamente un 2%. El discriminante cuadrático no se propondría debido a un sesgo excesivo y los otros clasificadores arrojan resultados similares y en general

peores. En cuanto a la normalidad, los tests parecen indicar que dos de las clases se acercarían a esta condición, pero está claro que no se cumple en la clase restante.

4.4.4 Aplicación a los Datos del Síndrome de Cushing

Se trata de datos tomados de [Aitchison 75] de pacientes con el síndrome de Cushing, que es un desorden asociado con una secreción excesiva de cortisol por la glándula suprarenal. Se reconocen tres tipos diferentes del síndrome que son el adenoma, hiperplasia bilateral y carcinoma, que hacen referencia a la posible causa del exceso de secreción, y sólo puede ser determinado histopatológicamente.

Las características medidas han sido los ritmos de excreción urinaria de los metabolitos esteroides tetrahidrocortisona y pregnanetriol, en 21 pacientes.

4.4.4.1 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {6, 10, 5};   /*en cada clase*/
      Número-Clases: 3;    /*nº de clases*/
      Número-Características: 2;              /*nº de características*/
      Test1-Normalidad-Mono: {0.4790, 0.4537, 0.4923};          /*probabilidades
posteriori en cada clase*/
      Test2-Normalidad-Mono: {0.0018, 0.1446, 0.0005};          /*probabilidades
posteriori en cada clase*/
      Test1-Normalidad-Multi: {0.0041, 0.0687, 0.0097};     /*valores P en cada clase*/
      Distancia-Bhattacharyya: 1.6483;      /*para dos clases*/
      Distancia-Bhattacharyya-1: 0.3827;    /*para dos clases*/
      Distancia-Bhattacharyya-2: 0.9665;    /*para dos clases*/
      Sesgo-Lineal: 0.0077;/*para dos clases*/
      Sesgo-Cuadrático: 0.0179;       /*para dos clases*/
      Dimensionalidad-Intrínseca: 3;
      Reducción-Dimensionalidad: ninguno   /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

Realmente se dispone de muy pocos datos, aunque las clases están bastante separadas, sobre todo en covarianzas. El error alcanzado con la mayoría de clasificadores se sitúa en torno al 4%.

4.4.5 Aplicación a los Datos de Diabetes en los Indios Pima

La población de la que se han tomado los datos está constituida por mujeres de al menos 21 años de edad de la etnia de los indios Pima que viven cerca de Phoenix en Arizona. En esta población se hicieron una serie de tests para comprobar si había problemas de diabetes, siguiendo las recomendaciones de la Organización Mundial de la Salud. Los datos recogidos están disponibles en [Murphy 95].

Las características medidas han sido: número de embarazos, concentración de glucosa en el plasma sanguíneo, presión arterial, grosor del pliegue de la piel del triceps, insulina en suero, índice de masa corporal, función de antecedentes familiares de diabetes, edad.

El número de datos recopilados es bastante grande, concretamente, 532 de los cuales 355 corresponden a casos de diabetes y 177 a mujeres que no tienen diabetes.

4.4.5.1 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {355, 177};  /*en cada clase*/
      Número-Clases: 2;     /*nº de clases*/
      Número-Características: 8;                /*nº de características*/
      Test1-Normalidad-Mono: {0.0000, 0.0000};              /*probabilidades posteriori en
cada clase*/
      Test2-Normalidad-Mono: {0.0000, 0.0000};              /*probabilidades posteriori en
cada clase*/
      Test1-Normalidad-Multi: {0.0000, 0.0000};      /*valores P en cada clase*/
      Distancia-Bhattacharyya: 0.0608;       /*para dos clases*/
      Distancia-Bhattacharyya-1: 0.0070;     /*para dos clases*/
      Distancia-Bhattacharyya-2: 0.0538;     /*para dos clases*/
```

```
        Sesgo-Lineal: 0.0453;/*para dos clases*/
        Sesgo-Cuadrático: 0.0825;        /*para dos clases*/
        Dimensionalidad-Intrínseca: 6;
        Reducción-Dimensionalidad: ninguno    /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

En este caso, el número de datos es bastante grande por lo que se pueden diseñar los clasificadores en mejores condiciones. El mínimo error que se obtuvo es de un 20% con el discriminante lineal y con el discriminante cuadrático se llegó a un 22% (el término de sesgo cuadrático es el doble del término de sesgo lineal). A pesar de todo, ninguno de estos clasificadores fue propuesto por el sistema, ya que, en principio las distancias de Bhattacharyya medidas no lo indican. Posiblemente existe algún tipo de simetría que explica esta circunstancia.

Como solución subóptima, en las redes neuronales entrenadas se alcanzó un 22%, con 24 neuronas en la capa oculta y parámetro de *Weight decay* igual a cero. El mejor resultado con el clasificador de vecinos próximos fue de un 23% con tres vecinos.

B. D. Ripley [Ripley 96] llega a un error del 20%, en el mejor de los casos.

4.4.6 Aplicación a Datos de Biopsia de Cáncer de Mama

En este caso se trata de resultados de 699 biopsias de tumores realizadas en hospitales de Wisconsin. De los 699 tumores analizados, 458 resultaron ser benignos y 241 malignos.

Se consideraron nueve variables, relevantes para el problema en cuestión.

4.4.6.1 Clasificadores Propuestos para estos Datos por el Sistema Basado en el Conocimiento

```
CONCEPT Información-Patrones;
   DESCRIPTION: "Información extraída de los patrones de entrada";
   ATTRIBUTES:
      Número-Patrones-Entrada: {458, 241};  /*en cada clase*/
      Número-Clases: 2;     /*nº de clases*/
      Número-Características: 9;                /*nº de características*/
```

```
        Test1-Normalidad-Mono: {0.0000, 0.0000};                /*probabilidades posteriori en
cada clase*/
        Test2-Normalidad-Mono: {0.0000, 0.0000};                /*probabilidades posteriori en
cada clase*/
        Test1-Normalidad-Multi: {0.0000, 0.0000};       /*valores P en cada clase*/
        Distancia-Bhattacharyya: 3.6860;        /*para dos clases*/
        Distancia-Bhattacharyya-1: 1.7064;      /*para dos clases*/
        Distancia-Bhattacharyya-2: 1.9796;      /*para dos clases*/
        Sesgo-Lineal: 0.0000;/*para dos clases*/
        Sesgo-Cuadrático: 0.0000;       /*para dos clases*/
        Dimensionalidad-Intrínseca: 5;
        Reducción-Dimensionalidad: ninguno    /*método reducción dimensionalidad*/
END CONCEPT Información-Patrones;
```

En este último ejemplo, disponemos también de gran cantidad de datos. La información obtenida de las muestras parece indicar que existe una separabilidad bastante grande tanto en medias como en covarianzas. Tanto con el discriminante lineal como con el cuadrático se alcanza un error del 6%. Las redes neuronales consiguen aproximarse al 0%, y resultan ser los mejores clasificadores. Con el clasificador de vecinos próximos (3 vecinos) se llega a un 5% de error.

CONCLUSIONES, APORTACIONES Y LÍNEAS ABIERTAS

Conclusiones

En este trabajo se han desarrollado una serie de herramientas de análisis y clasificación de datos que automatizan el proceso de diagnóstico. Estas herramientas se han integrado conjuntamente, mediante un Sistema Basado en el Conocimiento, diseñado siguiendo la metodología CommonKADS e implementado a través de una serie de librerías en lenguaje CLIPS.

El proceso de diagnóstico se ha contemplado como un proceso de diseño de clasificadores, que dependiendo de los datos disponibles y del conocimiento experto introducido en el sistema, propone el clasificador o clasificadores más adecuados, atendiendo a unos criterios preestablecidos.

Especial atención se ha dedicado al estudio de la normalidad de un conjunto de datos. Se han propuesto dos enfoques alternativos al clásico, basado en contrastes de significación con una sola hipótesis. El primero de los enfoques consistió en utilizar una red neuronal para combinar diferentes estadísticos, usados habitualmente en la

determinación de la normalidad. En la segunda aproximación al problema, se utilizó Teoría de las Grandes Desviaciones para seleccionar un conjunto reducido de distribuciones alternativas a la normalidad que permitieron diseñar un test informativo y eficiente. La filosofía de ambas técnicas está inspirada en la combinación de "expertos", cuya acción conjunta puede resultar más efectiva que la individual, y de hecho esto fue confirmado a través de los numerosos experimentos realizados.

En lo que se refiere a la validación del SBC con los datos de Alzheimer y Dislexia, se obtuvieron resultados bastante satisfactorios. Para el diagnóstico del Alzheimer, el error de clasificación logrado fue muy bajo (aproximadamente un 1%), incluso con clasificadores simples como los discriminantes lineal y cuadrático. En el caso de la dislexia, la cifra más baja de error se alcanzó con una red neuronal y con los clasificadores de vecinos próximos (aproximadamente un 25%).

Por último, es importante resaltar de nuevo la generalidad de los procedimientos empleados, que los hacen adaptables a otros problemas de diferente naturaleza.

Aportaciones

Después de las conclusiones generales anteriores, pasamos a concretar cuáles son las aportaciones principales de este trabajo.

- Proponer una arquitectura software de implementación del Modelo de Conocimiento de CommonKADS basada en una serie de librerías en CLIPS.
- Conservación de la entidad de los objetos genéricos y estructuras definidos en el Modelo de Conocimiento en la implementación.
- Creación de un traductor que automatiza parcialmente el proceso de conversión de objetos CommonKADS a CLIPS.
- Aplicación de lo expuesto en los tres puntos anteriores al dominio del diseño de clasificadores.
- Considerar el problema del diseño de un clasificador como un problema de diseño general, cuya resolución se basa en la disponibilidad de conocimiento explícito que sugiera estrategias para restringir el espacio de posibles diseños. De hecho, se ha utilizado el método general: Proponer-Criticar-Modificar, con algunas variantes para adaptarlo a nuestro problema particular. El conocimiento experto utilizado para diseñar el clasificador, se compone de heurísticas sencillas, obtenidas a partir de

pruebas experimentales realizadas por diversos investigadores del área y también a partir de nuestra propia experiencia.

- Enfocar el problema de verificación de la normalidad desde una perspectiva Bayesiana, proponiendo el sistema de distribuciones de Johnson como conjunto de alternativas válido para ser utilizado como base de las técnicas desarrolladas.
- Utilización de la red neuronal como sistema para la combinación de estadísticos, demostrando que sus salidas constituyen una buena aproximación a las probabilidades a posteriori de seguir la distribución normal. Además se pone de manifiesto su buen comportamiento (en términos de error promedio y potencia) al compararlo con los tests clásicos.
- Utilización de la Teoría de Grandes Desviaciones para determinar un número muy limitado de distribuciones alternativas que, conjuntamente, resultan equivalentes a considerar un número mucho mayor. Esto nos ha permitido diseñar un test más informativo acerca del tipo de no normalidad. También se ha comparado su eficiencia con los tests clásicos, encontrándose buenos resultados. Por último, se ha constatado que la aproximación asintótica en la que se fundamenta la Teoría de Grandes Desviaciones, sigue siendo razonablemente válida (en nuestro problema), incluso para tamaños de muestra N muy pequeños y por lo tanto alejados de la suposición $N \to \infty$.
- Aplicación de las técnicas desarrolladas a dos problemas particulares de gran interés: el diagnóstico del Alzheimer y el diagnóstico de Dislexias.

Líneas Abiertas

- Extender el análisis de la normalidad univariante a más dimensiones. El procedimiento pasaría por escoger una alternativa válida multidimensional para la normalidad, haciendo aplicables las técnicas propuestas en el Capítulo 3.
- Aprovechar la generalidad de estos métodos para su aplicación a otras hipótesis acerca de la forma de las distribuciones.
- Aplicar a nuevos problemas o dominios el Sistema Basado en Conocimiento para el diseño del clasificador, lo cual permitirá extender y validar nuevas estrategias.

- Completar la automatización del proceso de hacer operativo un Modelo de Conocimiento, integrando funcionalidades nuevas en el programa traductor como, por ejemplo, la sintaxis que permita describir la estructura de control dentro de las tareas.
- Crear un entorno visual de modelización de Conocimiento, donde, de una forma natural, se integren los catálogos o librerías de distintos componentes reusables CommonKADS junto con el traductor a CLIPS. De esta forma se crearía un entorno de trabajo que haría más ágil la reutilización de componentes y su conversión a ejecutables CLIPS, gracias al empleo de representaciones gráficas.

APÉNDICE

LISTADO 1: Librería CLIPS de Objetos Genéricos de CommonKADS

```
;                         ---             CONCEPT          ---
; Desde el punto de vista de commonKads:
; Conjunto de objetos o instancias que ocurren en el dominio de la aplicación y que
; comparten características similares.
; Desde el punto de vista de su implementación en Clips:
; Clase que sirve como base para definir cualquier concepto que pertenezca a un dominio
; de la aplicación. Para la definición de un concepto, se crea una nueva clase derivada
; de este concepto, que contendrá además sus características específicas.
;
(defclass CONCEPT (is-a USER) (role abstract)
        (slot description)       ;;; descripción del concepto
                                 ;;; ... al definir la subclase correspondiente
                                 ;;; ... a un concepto concreto se definen sus
                                 ;;; ... atributos con nuevos slots.
                                 ;;; ...
)

;                         ---             ARGUMENT         ---
; No tiene equivalente en la teroría de commonKads.
; Desde el punto de vista de su implementación en Clips:
; Clase para la especificación de los conceptos y su cardinalidad que intervienen en
; una determinada relación o ruletype (definidos más abajo). Se creará una instancia de
; ARGUMENT por cada argumento necesario, enlazandose introduciendo su nombre en el
; slot correspondiente dentro de la relación o ruletype que lo use.
;
(defclass ARGUMENT (is-a USER) (role concrete)
        (slot binding-name       ;;; nombre del concepto al que se refiere
                        (create-accessor read-write))
        (slot cardinality-min    ;;; Cardinalidad mínima
                        (create-accessor read-write))
```

```
        (slot cardinality-max    ;;; Cardinalidad máxima
                        (create-accessor read-write))
)

;                       ---             RELATION        ---
; Desde el punto de vista de commonKads:
; Define relaciones entre conceptos.
; Desde el punto de vista de su implementación en Clips:
; Clase que sirve como base para definir cualquier relación que pertenezca a un dominio
; de la aplicación. Para la definición de una relación, se crea una nueva clase derivada
; de esta clase, que contendrá además sus características específicas.
;
(defclass RELATION (is-a USER) (role abstract)
        (slot description)       ;;; descripcion de la relación
        (multislot arguments)    ;;; argumentos de la relación
        (multislot values)       ;;; instancias que cumplen la relación
                                        ;;; ...
                                        ;;; ... al definir la relación
                                        ;;; ... se definen sus
                                        ;;; ... atributos con nuevos slots.
                                        ;;; ...
)

;                       ---             RULETYPE        ---
; Desde el punto de vista de commonKads:
; Representa dependencias entre conceptos, sin llegar a especificar de forma concreta
; las reglas de ese tipo de dependencia, sino como deben aplicarse éstas.
; Desde el punto de vista de su implementación en Clips:
; No deben crearse objetos de esta clase. Para ello existen dos subclases que representan
; los dos posibles tipos a utilizar.
;
(defclass RULETYPE (is-a USER) (role abstract)
        (slot description        ;;; descripcion del ruletype
                        (create-accessor read-write))
)

;                       ---             CONSTRAINT-RULETYPE             ---
; Desde el punto de vista de commonKads:
; Ruletype donde la dependencia es una restricción sobre alguno o varios atributos del
; concepto.
; Desde el punto de vista de su implementación en Clips:
; Clase que sirve como base para definir cualquier constraint-ruletype que pertenezca a
; un dominio de la aplicación. Para la definición de un constraint-ruletype, se crea una
; instancia de esta clase, que contendrá en su slot <item> el nombre de una instancia de
; la clase ARGUMENT.
;
(defclass CONSTRAINT-RULETYPE (is-a RULETYPE) (role concrete)
        (slot item                      ;;; elemento del que se definen sus restricciones
                        (create-accessor read-write))
)

;                       ---             IMPLICATION-RULETYPE            ---
; Desde el punto de vista de commonKads:
; Ruletype donde la dependencia es una relación sobre alguno o varios atributos de
; dos conceptos (antecedente y consecuente).
; Desde el punto de vista de su implementación en Clips:
; Clase que sirve como base para definir cualquier implication-ruletype que pertenezca a
; un dominio de la aplicación. Para la definición de un implication-ruletype, se crea una
; instancia de esta clase, que contendrá en sus slot <antecedent> y <consequent> los nombres
; de una instancia de la clase ARGUMENT, y en su slot <connection-symbol> un nombre
; descriptivo del tipo de relación. Este último slot no es usado en la implementación, ya
; que en las reglas se indica a qúe ruletype pertence.
;
(defclass       IMPLICATION-RULETYPE (is-a RULETYPE) (role concrete)
        (multislot antecedent           ;;; elemento que formará el antecedente
                        (create-accessor read-write))
        (slot consequent         ;;; elemento que formará el consecuente
                        (create-accessor read-write))
        (slot connection-symbol  ;;; simbolo que define el ruletype
                        (create-accessor read-write))
)

;;;---------------------------- FIN componentes de los domain-schemas ------------------------


;;;---------------------------- componentes de las knowledge-bases ----------------------------

;                       ---             KNOWLEDGE-BASE          ---
; Desde el punto de vista de commonKads:
; Información específica acerca de los conocimientos definidos en los domain-schemas,
```

```
; tales como reglas concretas de un ruletype o instancias concretas de un concepto.
; Desde el punto de vista de su implementación en Clips:
; Se encarga de extraer información de la base de conocimientos a través de los
; message-handler que usan las inferencias.
;
(defclass KNOWLEDGE-BASE (is-a USER) (role concrete)
        (slot knowledge-base-name (create-accessor read-write))
)


; Función que es usada para extraer la información.
; Cada inferencia tiene un módulo individual que contendrá los hechos y reglas que usará,
; con lo que se evita la mezcla de conocimiento de distintas inferencias.
; Pone como entorno de trabajo el modulo de la inferencia que lo utiliza, elimina la
; información que ya no es necesaria producto de anteriores ejecuciones de esta inferencia.
; Activa las reglas necesarias las ejecuta (obteniendo la información necesaria para la
; inferencia) y vuelve a poner el entorno de trabajo global. Una base de conocimientos puede
; ser usada por muchas inferencias, por lo que deberemos pasar en la llamada que inferencia
; solicita información a la base de conocimientos (nombre del modulo de la inferencia).
(defmessage-handler KNOWLEDGE-BASE inferencing-function primary (?inference-module-name)
;;;     (printout t "realizando una inferencia en la base de conocimientos " (instance-name ?self) crlf)
        (focus ?inference-module-name)
        (retract *)
        (progn$ (?r (get-defrule-list ?inference-module-name)) (refresh  ?r))
        (run)
        (focus MAIN)
)

;                       ---             RULE                    ---
; Desde el punto de vista de commonKads:
; Son expresiones que dan reglas concretas de los implication-ruletype definidos
; en los domain-schemas.
; Desde el punto de vista de su implementación en Clips:
; Son objetos que tienen información de las reglas de los ruletypes, y son analizados
; en tiempo de ejecución para generar la información resultante de un implication-ruletype.
; Durante una inferencia son analizados todos los objetos rules que pertenecen a los
; ruletypes de las knowledge-bases necesarios para esa inferencia.
; Están fomado por el nombre de la knowledge-base y ruletype a la que pertenecen, y por un
; antecedente y un consecuente que serán enlaces a instancias de objetos ANTECEDENT
; y CONSEQUENT que se definen más abajo.
;
(defclass RULE (is-a USER) (role concrete) (pattern-match reactive)
        (slot knowledge-base-name (create-accessor read-write))
        (slot ruletype (create-accessor read-write))
        (slot antecedent (create-accessor read-write))
        (slot consequent (create-accessor read-write))
)

;                       ---             ANTECEDENT                      ---
; Desde el punto de vista de commonKads:
; Son parte de las expresiones (antecedentes) que dan reglas concretas de los
; implication-ruletype definidos en los domain-schemas.
; Desde el punto de vista de su implementación en Clips:
; Son objetos que describen el antecedente de una regla. Cuando el antecedente no sea
; una expresión simple (con un sólo operador) una instancia del objeto ANTECEDENT se enlazará
; a su vez con otras instancias de este objeto que generarán las expresiones complejas
; en forma de árbol binario.
; En caso de que operator sea Or o AND left y right serán instancias de objetos ANTECEDENT,
; en caso contrario serán valores (numéricos o de atributos).
;
(defclass ANTECEDENT (is-a USER) (role concrete) (pattern-match reactive)
        (slot operator (create-accessor read-write))
        (slot left-operand (create-accessor read-write))
        (slot right-operand (create-accessor read-write))
)

;                       ---             CONSEQUENT                      ---
; Desde el punto de vista de commonKads:
; Son parte de las expresiones (consecuentes) que dan reglas concretas de los
; implication-ruletype definidos en los domain-schemas.
; Desde el punto de vista de su implementación en Clips:
; Son objetos que describen el consecuente de una regla. Cuando el consecuente no sea
; una expresión simple (con un sólo operador) una instancia del objeto CONSEQUENT se enlazará
; a su vez con otras instancias de este objeto que generarán las expresiones complejas
; en forma de árbol binario.
; En caso de que operator sea Or o AND left y right serán instancias de objetos CONSEQUENT,
; en caso contrario serán valores (numéricos o de atributos). Para diferenciar una condición
; de igualdad de una de asignación, es decir, cuando se usa un atributo en una expresión,
; distinguiremos cuando nos referimos al valor del atributo o a su nombre añadiendo a éste
; último los caracteres @@ .
```

```
; Ej: atributo = atributo + 2  ,se introduciría  atributo@@ = atributo + 2
;
(defclass CONSEQUENT (is-a USER) (role concrete) (pattern-match reactive)
        (slot operator (create-accessor read-write))
        (slot left-operand (create-accessor read-write))
        (slot right-operand (create-accessor read-write))
)

;;;---------------------------- FIN componentes de las knowledge-bases ------------------------


;;;---------------------------- componentes de los inference-knowledges ----------------------------

;                         ---             ROLE            ---
; Desde el punto de vista de commonKads:
; Son nombres abstractos de objetos de datos que indican su rol en el proceso de razonamiento.
; Desde el punto de vista de su implementación en Clips:
; Es una clase que sirve como base para la definición de STATIC-ROLE y DINAMIC-ROLE y que
; contendrá el enlace al objeto al que actua bajo ese rol y la función para acceder a los
; atributos de estos objetos.
;
(defclass ROLE (is-a USER) (role abstract)
        (slot data-type
                (create-accessor read-write))
        (multislot domain-mapping;;; multi es para permitir definiciones de ruletypes
                        (create-accessor read-write))
        (multislot attribute-mapping
                (create-accessor read-write))
)

; Función para leer objetos representados por este rol.
; El atributo deseado se debe pasar como parámetro.
(defmessage-handler ROLE read primary ()
;;;     (printout t "accediendo al rol actual " (instance-name ?self) crlf)
        (symbol-to-instance-name (nth$ 1 (send ?self get-domain-mapping)))
)

; Función para leer el valor de un atributo de los objetos representados por este rol.
; El atributo deseado se debe pasar como parámetro.
; A diferencia de read-attribute, no tiene en cuenta el mapeo de atributos.
(defmessage-handler ROLE direct-read-attribute primary (?atributo)
;;;     (printout t "accediendo al rol actual " (instance-name ?self) crlf)
        (bind ?get-atributo (sym-cat get- ?atributo))
        (send (symbol-to-instance-name (nth$ 1 (send ?self get-domain-mapping))) ?get-atributo)
)

; Función para leer un atributo de los objetos representados por este rol.
; El atributo deseado se debe pasar como parámetro.
(defmessage-handler ROLE read-attribute primary (?atributo)
;;;     (printout t "accediendo al rol actual " (instance-name ?self) crlf)
;;;                                             (nth$ (member$ (nth$ ?index ?lista-mapping) ?lista-
concepto) ?lista-rol)
        (bind ?get-atributo
                (sym-cat get-
                        (nth$
                                (member$ ?atributo (rest$ (class-slots (send ?self get-data-type)
inherit)))
                                (send ?self get-attribute-mapping)
                        )
                )
        )
        (send (symbol-to-instance-name (nth$ 1 (send ?self get-domain-mapping))) ?get-atributo)
)

;                         ---             STATIC-ROLE             ---
; Desde el punto de vista de commonKads:
; Son roles estables a lo largo del tiempo, es decir, el objeto que actúa bajo este rol
; será el mismo durante toda la ejecución. Especifica la colección de conocimiento del
; dominio que es usada para hacer la inferencia.
; Desde el punto de vista de su implementación en Clips:
; Es una clase derivada de la clase ROLE que no necesita ningún atributo más para su
; funcionalidad. Se define de esta forma para diferenciar los roles estáticos y dinámicos.
;
(defclass STATIC-ROLE (is-a ROLE) (role concrete)
)

; Devuelve el nombre de la instancia a la que hace referencia este static role.
(defmessage-handler STATIC-ROLE instance-name primary ()
        (nth$ 1 (send ?self get-domain-mapping))
)
```

```
; Devuelve el nombre del ruletype al que hace referencia este static role.
(defmessage-handler STATIC-ROLE ruletype-name primary ()
        (nth$ 1 (send ?self get-domain-mapping))
)

; Devuelve el nombre de la base de conocimientos a la que hace referencia este static role.
(defmessage-handler STATIC-ROLE knowledge-base-name primary ()
        (nth$ 3 (send ?self get-domain-mapping))
)

;                         ---              DINAMIC-ROLE             ---
; Desde el punto de vista de commonKads:
; Son entradas y salidas en tiempo de ejecución de las inferencias. En cada ejecución de
; la inferencia se pueden tener distintas instanciaciones del mismo rol, es decir
; un rol dinámico puede ser llevado a cabo por distintos objetos.
; Desde el punto de vista de su implementación en Clips:
; Es una clase derivada de la clase ROLE a la que se le añaden los atributos data-type
; para indicar el tipo de objeto que puede representar este rol y una función, para
; cambiar el objeto que representará el rol.
;
(defclass DYNAMIC-ROLE (is-a ROLE) (role concrete)
)

; Función para asignar un valor a un atributo de los objetos representados por este rol.
; El atributo deseado y su valor se debe pasar como parámetro.
; A diferencia de write-attribute, no tiene en cuenta el mapeo de atributos.
(defmessage-handler DYNAMIC-ROLE direct-write-attribute primary (?atributo $?valor)
;;;     (printout t "accediendo al rol actual " (instance-name ?self) crlf)
        (bind ?put-atributo (sym-cat put- ?atributo))
        (send (symbol-to-instance-name (nth$ 1 (send ?self get-domain-mapping))) ?put-atributo $?valor)
)

; Función para asignar un valor a un atributo de los objetos representados por este rol.
; El atributo deseado y su valor se debe pasar como parámetro.
(defmessage-handler DYNAMIC-ROLE write-attribute primary (?atributo $?valor)
;;;     (printout t "accediendo al rol actual " (instance-name ?self) crlf)
;;;     (bind ?put-atributo (sym-cat put- ?atributo))
        (bind ?put-atributo
                (sym-cat put-
                        (nth$
                                (member$ ?atributo (rest$ (class-slots (send ?self get-data-type)
inherit)))
                                (send ?self get-attribute-mapping)
                        )
                )
        )
        (send (symbol-to-instance-name (nth$ 1 (send ?self get-domain-mapping))) ?put-atributo $?valor)
)

; Función para cambiar el objeto que actúa con este rol. El nombre de la instancia
; del objeto deberá ser pasada como parámetro. Si
(defmessage-handler DYNAMIC-ROLE played-by primary ($?name-instance)
;;; (printout t "actualizando el rol actual " (instance-name ?self) crlf)
(if (neq nil (nth$ 1 $?name-instance)) then
        (bind ?lista-mapping (rest$ (class-slots (send ?self get-data-type) inherit)))
        (bind ?lista-rol (create$))
        (bind ?lista-concepto (create$))
        (if (eq (nth$ 1 $?name-instance) new-instance) then
                (send ?self put-domain-mapping (sym-cat (instance-name (make-instance of (nth$ 2 $?name-
instance)))))
                (bind ?lista-atributos (list (rest$ (rest$ $?name-instance))))
                (loop-for-count (?index 1 (list-length ?lista-atributos))
                        (if (eq into (list-nth 2 (list-nth ?index ?lista-atributos))) then
;;;                             (printout t "atributo mapeado" crlf)
                                (bind ?lista-rol (create$
                                                ?lista-rol
                                                (list-nth 1 (list-nth ?index ?lista-atributos))
                                ))
                                (bind ?lista-concepto (create$
                                                ?lista-concepto
                                                (list-nth 3 (list-nth ?index ?lista-atributos))
                                ))
                                ; asignar valor al atributo del concepto
                                (send ?self direct-write-attribute (list-first (list-nth ?index ?lista-
atributos))
                                                                (list2multifield (list-rest (list-rest (list-
rest (list-nth ?index ?lista-atributos))))))
                        else
;;;                             (printout t "atributo sin mapear" crlf)
```

```
                                (send ?self direct-write-attribute (list-first (list-nth ?index ?lista-
atributos))
                                                                (list2multifield (list-rest (list-nth ?index
?lista-atributos))))
                        )
                )
                (if (or (superclassp (send ?self get-data-type) (nth$ 2 $?name-instance))
                                                (eq (send ?self get-data-type) (nth$ 2 $?name-
instance))) then
                        (send ?self put-attribute-mapping (rest$ (class-slots (send ?self get-data-
type) inherit)))
                else
                        (send ?self put-attribute-mapping (create$))
                        (loop-for-count (?index 1 (length ?lista-mapping))
                                (if (member$ (nth$ ?index ?lista-mapping) ?lista-concepto) then
                                        (send ?self put-attribute-mapping (create$
                                                (send ?self get-attribute-mapping)
                                                (nth$ (member$ (nth$ ?index ?lista-mapping) ?lista-
concepto) ?lista-rol)
                                        ))
                                else
                                        (send ?self put-attribute-mapping (create$
                                                (send ?self get-attribute-mapping)
                                                NULL
                                        ))
                                )
                        )
                )
        else
                (send ?self put-domain-mapping (nth$ 1 $?name-instance))

                (bind ?concepto (class (symbol-to-instance-name (nth$ 1 $?name-instance))))

                (if (or (superclassp (send ?self get-data-type) ?concepto)
                                (eq (send ?self get-data-type) ?concepto)) then
                        (send ?self put-attribute-mapping ?lista-mapping)
                else
                        ;lista de atributos del rol a ser mapeados
                        (bind ?lista-rol (list2multifield (list-nth 1 (list (rest$ $?name-instance)))))
                        ;lista de atributos del concepto a mapear en el rol
                        (bind ?lista-concepto (list2multifield (list-nth 3 (list (rest$ $?name-
instance)))))
                        ;mapeo:
                        (send ?self put-attribute-mapping (create$))
                        (loop-for-count (?index 1 (length ?lista-mapping))
                                (if (member$ (nth$ ?index ?lista-mapping) ?lista-concepto) then
                                        (send ?self put-attribute-mapping (create$
                                                (send ?self get-attribute-mapping)
                                                (nth$ (member$ (nth$ ?index ?lista-mapping) ?lista-
concepto) ?lista-rol)
                                        ))
                                else
                                        (send ?self put-attribute-mapping (create$
                                                (send ?self get-attribute-mapping)
                                                NULL
                                        ))
                                )
                        )
                )
        )
;;;     (printout t "su attribute-mapping: " (send ?self get-attribute-mapping) crlf)
else
;;;     (printout t "era nil " (send ?self get-attribute-mapping) crlf)

)
)

(defmessage-handler DYNAMIC-ROLE assign primary ($?name-instance)
;;;(printout t "actualizando el rol actual " (instance-name ?self) crlf)
(bind ?source-role (nth$ 1 $?name-instance))
; si se especifica un rol y tiene un mapping valido ...
(if (and (neq nil ?source-role) (neq nil (send ?source-role get-domain-mapping)))then
        ; si no se especifica mapeo ...
        (if (eq nil (nth$ 1 (rest$ $?name-instance))) then
;;;             (printout t "Asignando " ?source-role " a " ?self crlf)
                (send ?self put-domain-mapping (send ?source-role get-domain-mapping))
                ; si los dos roles tienen el mismo tipo ...
                (if (or (superclassp (send ?self get-data-type) (send (symbol-to-instance-name ?source-
role) get-data-type))
```

```
                                        (eq (send ?self get-data-type) (send (symbol-to-instance-name
?source-role) get-data-type))) then
                        (send ?self put-attribute-mapping (send (symbol-to-instance-name ?source-role)
get-attribute-mapping))
                ; si el tipo es diferente ...
                else
                        (send ?self played-by (send (symbol-to-instance-name ?source-role) get-domain-
mapping))
                )
;;;             (printout t "su attribute-mapping: " (send ?self get-attribute-mapping) crlf)
        ; si se especifica mapeo ...
        else
                (send ?self put-domain-mapping (send ?source-role get-domain-mapping))
                (bind ?lista-mapping (rest$ (class-slots (send ?self get-data-type) inherit)))
                (bind ?lista-mapping-source (rest$ (class-slots (send ?source-role get-data-type)
inherit)))
                ;lista de atributos del rol a ser mapeados
                (bind ?lista-source (list2multifield (list-nth 1 (list (rest$ $?name-instance)))))
                ;lista de atributos del concepto a mapear en el rol
                (bind ?lista-concepto (list2multifield (list-nth 3 (list (rest$ $?name-instance)))))
                (bind ?lista-att-map (send ?source-role get-attribute-mapping))
;;;             (bind ?concepto (class (symbol-to-instance-name (nth$ 1 $?name-instance))))
                (bind ?concepto (send ?source-role get-data-type))
                ;mapeo:
                (send ?self put-attribute-mapping (create$))
                (loop-for-count (?index 1 (length ?lista-mapping))
                        (if (member$ (nth$ ?index ?lista-mapping) ?lista-concepto) then
                                (send ?self put-attribute-mapping (create$
                                                (send ?self get-attribute-mapping)
                                                (nth$ (member$ (nth$ (member$ (nth$ ?index ?lista-
mapping) ?lista-concepto) ?lista-source) ?lista-mapping-source) ?lista-att-map)
                                ))
                        else
                                (send ?self put-attribute-mapping (create$
                                                        (send ?self get-attribute-mapping)
                                                        NULL
                                ))
                        )
                )
;;;             (printout t "su attribute-mapping: " (send ?self get-attribute-mapping) crlf)
        )
else
;;;     (printout t "era nil " (send ?self get-attribute-mapping) crlf)
)
)


;                       ---             INFERENCE               ---
; Desde el punto de vista de commonKads:
; Describe el nivel más bajo de descomposición funcional, lleva a cabo un paso del
; razanamiento. El razonamiento puede ser imperativo o declarativo.
; Desde el punto de vista de su implementación en Clips:
; Objeto que representará una inferencia, consta de los roles de entrada, estáticos
; y de salida, del método que se usará para resolver esta inferencia y una cola con
; las soluciones deducidas a partir de la inferencia.
;
(defclass INFERENCE (is-a USER) (role concrete) (pattern-match reactive)
        (slot specification
                        (create-accessor read-write))
        (multislot input-roles
                        (create-accessor read-write))
        (multislot output-roles
                        (create-accessor read-write))
        (multislot static-roles
                        (create-accessor read-write))
        (multislot solutions-queue
                        (create-accessor read-write))
        (slot method
                        (create-accessor read-write))
        (slot has-solution-method
                        (create-accessor read-write))
        (slot new-solution-method
                        (create-accessor read-write))
)

; Esta función manda a ejecutar la inferencia mediante el método asociado a ella.
; Para ello le pasamos su nombre por parámetro de forma que el método pueda obtener
; los roles y la cola de soluciones de la inferencia que lo usa.
(defmessage-handler INFERENCE execute primary ($?roles)
;;;     (printout t "se esta ejecutando la inferencia " ?self crlf)
```

```
        (bind ?lista-roles (list $?roles))

        (loop-for-count (?index 1 (list-length ?lista-roles)) do
                (if (and (eq <-- (nth$ 3 (list-nth ?index ?lista-roles))) (neq ] (nth$ 4 (list-nth
?index ?lista-roles)))) then
                        (send (send ?self read-input-role (member$ (nth$ 2 (list-nth ?index ?lista-
roles)) (send ?self get-input-roles))) assign (nth$ 4 (list-nth ?index ?lista-roles)))
                )
        )

        (bind ?method-name (dynamic-get method))
        (bind ?comando (str-cat "(" ?method-name " [" (instance-name ?self) "])"))
        (eval ?comando)

        (loop-for-count (?index 1 (list-length ?lista-roles)) do
                (if (and (eq --> (nth$ 3 (list-nth ?index ?lista-roles))) (neq ] (nth$ 4 (list-nth
?index ?lista-roles)))) then
                        (send (nth$ 4 (list-nth ?index ?lista-roles)) assign (send ?self read-output-
role (member$ (nth$ 2 (list-nth ?index ?lista-roles)) (send ?self get-output-roles))))
                )
        )

)

; Necesario para conocer si la inferncia que se ha ejecutado tiene solución. Para ello
; hace una llamada a una funcion <nombre-inferencia>-has-solution, que debe ser
; implementada por el usuario.
(defmessage-handler INFERENCE has-solution primary ()
;;;     (printout t "miramos si " (instance-name ?self) " tiene solución" crlf)
        (bind ?has-solution-method-name (dynamic-get has-solution-method))
        (bind ?comando (str-cat "(" ?has-solution-method-name " [" (instance-name ?self) "])"))
        (eval ?comando)
)

; Necesario para obtener una nueva solución después de que la inferncia se haya ejecutado.
; Para ello hace una llamada a una funcion <nombre-inferencia>-new-solution, que debe ser
; implementada por el usuario.
(defmessage-handler INFERENCE new-solution primary ()
;;;     (printout t "buscamos nueva solución para " (instance-name ?self) crlf)
        (bind ?new-solution-method-name (dynamic-get new-solution-method))
        (bind ?comando (str-cat "(" ?new-solution-method-name " [" (instance-name ?self) "])"))
        (eval ?comando)
)

; Necesario para introducir una instancia que forma parte de las posibles soluciones de
; esta inferencia, a la cola de soluciones de la misma.
(defmessage-handler INFERENCE add-solutions-queue primary (?new-solution ?position)
        (if (eq ?position fin) then
                (send ?self put-solutions-queue (create$ ?self:solutions-queue ?new-solution))
        else
                (send ?self put-solutions-queue (insert$ ?self:solutions-queue ?position ?new-solution))
        )
)

; Devuelve el input rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-input-role (?posicion)
        (symbol-to-instance-name (nth$ ?posicion (send ?self get-input-roles)))
)

; Devuelve el atributo <atributo> del input rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-input-role-attribute (?posicion ?atributo)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-input-roles))) read-attribute
?atributo)
)

; Devuelve el static rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-static-role (?posicion)
        (symbol-to-instance-name (nth$ ?posicion (send ?self get-static-roles)))
)

; Devuelve el atributo <atributo> del static rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-static-role-attribute (?posicion ?atributo)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-static-roles))) read-attribute
?atributo)
)

; Devuelve el output rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-output-role (?posicion)
        (symbol-to-instance-name (nth$ ?posicion (send ?self get-output-roles)))
```

```
)

; Devuelve el atributo <atributo> del output rol situado en la posición <posicion>.
(defmessage-handler INFERENCE read-output-role-attribute (?posicion ?atributo)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-output-roles))) read-attribute
?atributo)
)

; Escribe el atributo <atributo> del output rol situado en la posición <posicion> con los
; valores <values>.
(defmessage-handler INFERENCE write-output-role-attribute (?posicion ?atributo $?values)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-output-roles))) write-attribute
?atributo $?values)
)

;;;;not yet
(defclass TRANSFER-FUNCTION (is-a USER) (role concrete)
        (slot I/O-roles)
)

;;;---------------------------- FIN componentes de los inference-knowledges  ------------------------


;;;---------------------------- componentes de los task-knowledges ----------------------------

;                        ---             TASK                    ---
; Desde el punto de vista de commonKads:
; Define una función compleja de razonamiento.
; Desde el punto de vista de su implementación en Clips:
; Objeto que representará un task. En él se definen los roles de entrada y salida,
; del método que se usará para resolver esta tarea y el objetivo de la tarea (una descripción
; de cual es el resultado perseguido).
;
(defclass TASK (is-a USER) (role concrete)
        (multislot I-roles              ;;; roles de entrada
                        (create-accessor read-write))
        (multislot O-roles              ;;; roles de salida
                        (create-accessor read-write))
        (slot goal                              ;;; comentario del objetivo
                        (create-accessor read-write))
        (slot method                    ;;; metodo de resolucion de la tarea
                        (create-accessor read-write))

)

; Esta función manda a ejecutar la tarea mediante el método asociado a ella.
; Para ello le pasamos los atributos por parámetro.
(defmessage-handler TASK execute primary ()
;;;     (printout t "se esta ejecutando esta tarea" crlf)
        (bind ?method-name (dynamic-get method))
        (bind ?comando (str-cat "(" ?method-name " [" (instance-name ?self) "])"))
        (eval ?comando)
)

; Devuelve el input rol situado en la posición <posicion>.
(defmessage-handler TASK read-input-role (?posicion)
        (symbol-to-instance-name (nth$ ?posicion (send ?self get-I-roles)))
)

; Devuelve el atributo <atributo> del input rol situado en la posición <posicion>.
(defmessage-handler TASK read-input-role-attribute (?posicion ?atributo)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-I-roles))) read-attribute
?atributo)
)

; Devuelve el output rol situado en la posición <posicion>.
(defmessage-handler TASK read-output-role (?posicion)
        (symbol-to-instance-name (nth$ ?posicion (send ?self get-O-roles)))
)

; Devuelve el atributo <atributo> del output rol situado en la posición <posicion>.
(defmessage-handler TASK read-output-role-attribute (?posicion ?atributo)
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-O-roles))) read-attribute
?atributo)
)

; Escribe el atributo <atributo> del output rol situado en la posición <posicion> con los
; valores <values>.
(defmessage-handler TASK write-output-role-attribute (?posicion ?atributo $?values)
```

```
        (send (symbol-to-instance-name (nth$ ?posicion (send ?self get-O-roles))) write ?atributo
$?values)
)
```

LISTADO 2: Código MATLAB para Distribuciones de Johnson

```
        function y=johnpdf(x,media,devstd,sqrtb1,b2)

%function y=johnpdf(x,media,devstd,sqrtb1,b2)
%
%evalua la funcion densidad de Johnson correspondiente en los x especificados
%x: vector con los puntos a evaluar
%media: media de la distribucion
%devstd: desviacion tipica de la distribucion
%sqrtb1: sesgo
%b2: kurtosis

parametros=johnson(media,devstd,sqrtb1,b2);
xi=parametros(1);
xlam=parametros(2);
gamma=parametros(3);
delta=parametros(4);
tipo=parametros(5);
ifault=parametros(6);

if ifault == 1
        disp('error: desviacion tipica menor que cero');
        return;
end

if ifault == 2 | tipo == 5
        px=nan*x;        %puntos aproximadamente en region b2<sqrtb1^2+1
        y=px;
        return;
end

if tipo == 1
        %Tipo 1 (Sl)
        y=(x-xi)./xlam;
        cero=(y<=0);
        xnul=cero;
        y(xnul)=.5;
        px=(delta./(sqrt(2.*pi).*abs(xlam).*y)).*(exp((-1/2).*(gamma+delta.*log(y)).^2));
        px(xnul)=0;
elseif tipo == 2
        %Tipo 2 (Su)
        y=(x-xi)./xlam;
        px=(delta./(sqrt(2.*pi).*abs(xlam).*sqrt(y.^2+1))).*(exp((-
1/2).*(gamma+delta.*log(y+sqrt(y.^2+1))).^2));
elseif tipo == 3
        %Tipo 3 (Sb)
        y=(x-xi)./xlam;
        cero=y<=0;
        uno=y>=1;
        xnul=cero | uno;
        y(xnul)=.5;      %para que no de warning al calcular la densidad
        px=(delta./(sqrt(2.*pi).*abs(xlam).*y.*(1-y))).*(exp((-1/2).*(gamma+delta.*log(y./(1-y))).^2));

        px(xnul)=0;
else
        %Tipo 4 (Normal)
        mu=media;
        sigma=devstd;
        %px=(1./(sigma.*sqrt(2.*pi))).*(exp(-((x-mu).^2)/(2.*sigma.^2)));
        px=normpdf(x,mu,sigma);

end

y=px;
```

LISTADO 3: Código MATLAB para Distribuciones de Johnson

```
        function y=johngen(N,M,tipo,xi,xlam,gamma,delta)

%y=johngen(N,M,tipo,xi,xlam,gamma,delta)
%
%genera una matriz de muestras N*M con M muestras de tamaño N
%correspondientes a la distribucion de Johnson de parametros especificados
%N: tamaño de la muestra
%M: numero de muestras
%tipo: indicador del tipo de distribucion de Johnson
%xi: parametro para obtener la media deseada
%xlam: parametro para obtener la desviacion tipica deseada
%gamma: parametro gamma
%delta: parametro delta
%ver johnson.m
%ver rutina 100 en lib.stat.cmu.edu

muestras=normrnd(0,1,N,M);

if tipo == 4     %distribucion normal
        muestras=(muestras-gamma)/delta;
        y=muestras;
elseif tipo == 1 %distribucion lognormal Sl
        %le aplicamos la transformacion para convertirlas a lognormal
        muestras=xlam.*exp((xlam.*muestras-gamma)./delta)+xi;
        y=muestras;
elseif tipo == 2 %distribucion Su
        %le aplicamos la transformacion para convertirlas a Su
        w=exp((muestras-gamma)/delta);
        w=(1/2)*(w-1./w);
        muestras=xlam*w+xi;
        y=muestras;
elseif tipo == 3 %distribucion Sb
        %le aplicamos la transformacion para convertirlas a Su
        w=(muestras-gamma)./delta;
        v=exp(-abs(w));
        v=(1-v)./(1+v);
        muestras=(1/2)*xlam*(sign(w).*v+1)+xi;
        y=muestras;
else
        %disp('Error: el tipo debe ser 0, 1, 2 o 3');
        y=muestras*0;
end
```

LISTADO 4: Código MATLAB para Distribuciones de Pearson

```
        function y=pearpdf(x,sqrbeta1,beta2)

%function y=pearpdf(x,sqrbeta1,beta2)
%
%evalua la funcion densidad de Pearson correspondiente en los x especificados
%x: vector con los puntos a evaluar
%sqrbeta1: parametro de sesgo (Fisher skewness=mu3/sigma^3)
%beta2: parametro de kurtosis (Pearson kurtosis=mu4/sigma^4)
%ver pag. 216 del Kendall

beta1=sqrbeta1^2;
tipo=peartype(sqrbeta1,beta2);    %determinamos el tipo de Pearson
if tipo == 0
        y=normpdf(x,0,1);         %normal
elseif tipo == 1
        %p y q para beta de primera clase
        g1=beta1;
        g2=beta2;
```

```
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        q=p;
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=(3*(-1-g1+g2+q))/(6+3*g1-2*g2);
        y=betapdf(x,p,q);
        cero=x<0;
        uno=x>1;
        xnul=cero | uno;
        y(xnul)=0;       %se ponen a cero los puntos en los que no esta definida
elseif tipo == 2         %simetrica (se calcula como caso limite de la beta)
        g1=beta1;
        g2=beta2;
        g1=g1+.000000001;
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        q=p;
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=(3*(-1-g1+g2+q))/(6+3*g1-2*g2);
        y=betapdf(x,p,q);
        cero=x<0;
        uno=x>1;
        xnul=cero | uno;
        y(xnul)=0;       %se ponen a cero los puntos en los que no esta definida
elseif tipo == 3 %gamma
        a=6/(beta2-3);
        b=1;
        y=gampdf(x,a,b);
        cero=x<0;
        y(cero)=0;
elseif tipo == 4
        %el tipo IV hay que tratarlo aparte, solo se deja el espacio
        y=zeros(size(x,1),size(x,2));
elseif tipo == 5
        p=(4*(2+beta1+sqrt(4+beta1)))/beta1;
        gam=1;
        y=((gam.^(p-1))/(gamma(p-1))).*(x.^(-p)).*exp(-gam./x);
        cero=x<0;
        y(cero)=0;
elseif tipo == 6
        %p y q para beta de segunda clase (Pearson tipo VI)
        g1=beta1;
        g2=beta2;
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=4-((3*(4+g1))/(6+3*g1-2*g2));
        y=(1./beta(p,q)).*((x.^(p-1))./((1+x).^(p+q)));
        cero=x<0;
        y(cero)=0;
elseif tipo == 7 %Student's t
        r=(6/(beta2-3))+4;
        y=tpdf(x,r);
else
        y=[];
end
```

LISTADO 5: Código MATLAB para Distribuciones de Pearson

```
        function y=peartype(sqrbeta1,beta2)

%function y=peartype(sqrbeta1,beta2)
%
%determina el tipo de la distribucion de Pearson de parametros sqrbeta1 y beta2
%sqrbeta1: parametro de sesgo (Fisher skewness=mu3/sigma^3)
%beta2: parametro de kurtosis (Pearson kurtosis=mu4/sigma^4)
%ver pag. 216 del Kendall

beta1=sqrbeta1^2;
if beta2 <= (beta1 +1)
        disp('error: excedido limite superior para toda distribucion');
        tipo=inf;
```

```
elseif beta2 >= (15/8)*beta1 +(36/8)
        disp('error: excedido limite inferior para distribucion Pearson');
        tipo=inf;
elseif beta1 == 0 & beta2 < 3
        tipo=2;  %Pearson tipo II
elseif beta1 == 0 & beta2 > 3
        tipo=7; %Pearson tipo VII (Student's t-distribution)
elseif beta1 == 0 & beta2 == 3
        tipo=0; %Normal
elseif beta2 > (beta1+1) & beta2 < (3+1.5*beta1)
        tipo=1;  %Pearson tipo I (beta de 1ª clase)
elseif beta2 == (3+1.5*beta1)
        tipo=3;  %Pearson tipo III (gamma)
elseif beta2 > (3*(-16-13*beta1-2*(4+beta1)^(3/2)))/(-32+beta1) & beta2 < (15/8)*beta1 +(36/8)
        tipo=4;  %Pearson tipo IV
elseif beta2 == (3*(-16-13*beta1-2*(4+beta1)^(3/2)))/(-32+beta1)
        tipo=5;  %Pearson tipo V
elseif beta2 < (3*(-16-13*beta1-2*(4+beta1)^(3/2)))/(-32+beta1) & beta2 > (3+1.5*beta1)
        tipo=6;  %Pearson tipo VI
else
        disp('tipo desconocido');
end
y=tipo;
```

LISTADO 6: Código MATLAB para Distribuciones de Pearson

```
        function y=peargen(sqrbeta1,beta2,N,M)

%function y=peargen(sqrbeta1,beta2,N,M)
%
%genera una matriz de muestras N*M con M muestras de tamaño N
%correspondientes a la distribucion de Pearson de parametros beta1 y beta2
%la media y la desviacion tipica de la muestra dependera del tipo
%sqrbeta1: parametro de sesgo (Fisher skewness=mu3/sigma^3)
%beta2: parametro de kurtosis (Pearson kurtosis=mu4/sigma^4)
%N: tamaño de la muestra
%M: numero de muestras
%ver pag. 216 del Kendall

beta1=sqrbeta1^2;
tipo=peartype(sqrbeta1,beta2);    %determinamos el tipo de Pearson
if tipo == 0
        muestra=normrnd(0,1,N,M);%normal
elseif tipo == 1
        %p y q para beta de primera clase
        g1=beta1;
        g2=beta2;
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        q=p;
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=(3*(-1-g1+g2+q))/(6+3*g1-2*g2);
        muestra=betarnd(p,q,N,M);
elseif tipo == 2         %simetrica (se calcula como caso limite de la beta)
        g1=beta1;
        g2=beta2;
        g1=g1+.000000001;
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        q=p;
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=(3*(-1-g1+g2+q))/(6+3*g1-2*g2);
        muestra=betarnd(p,q,N,M);
elseif tipo == 3 %gamma
        a=6/(beta2-3);
        b=1;
        muestra=gamrnd(a,b,N,M);
elseif tipo == 4
        %el tipo IV hay que tratarlo aparte, solo se deja el espacio
        muestra=zeros(N,M);
elseif tipo == 5
        %se puede obtener a partir de la gamma con y=(1/x)
```

```
        p=(4*(2+beta1+sqrt(4+beta1)))/beta1;
        gam=1;
        a=p-1;
        b=1/gam;
        muestra=gamrnd(a,b,N,M);
        muestra=1./muestra;
elseif tipo == 6
        %p y q para beta de segunda clase (Pearson tipo VI)
        g1=beta1;
        g2=beta2;
        p=(g1*((1+g1-g2)^2)*((3+g2)^2))/(sqrt(-g1*((1+g1-g2)^2)*((3+g2)^2)*(36*(g1^2)+32*(-3+g2)*g2-
g1*(-63+g2*(78+g2)))));
        p=(3*(-1-g1+g2-p))/(6+3*g1-2*g2);
        q=4-((3*(4+g1))/(6+3*g1-2*g2));
        muestra=betarnd(p,q,N,M);
        muestra=muestra./(1-muestra);     %se hace el cambio de variable
                                          %de beta 1ª a beta 2ª
elseif tipo == 7 %Student's t
        r=(6/(beta2-3))+4;
        muestra=trnd(r,N,M);
else
end

if sqrbeta1 > 0
        y=muestra;
else
        %para que el sesgo sea negativo
        param=pearmom(sqrbeta1,beta2);
        y=param(1)-muestra;
end
```

LISTADO 7: Código MATLAB para Distribuciones de Pearson

```
        function y=ivpdf(x,sqrbeta1,beta2,k)

%function y=ivpdf(x,sqrbeta1,beta2,k)
%
%evalua la funcion densidad de Pearson IV correspondiente en los x especificados
%x: vector con los puntos a evaluar
%sqrbeta1: parametro de sesgo (Fisher skewness=mu3/sigma^3)
%beta2: parametro de kurtosis (Pearson kurtosis=mu4/sigma^4)
%k: constante de normalizacion para que la integral sea 1
%ver pag. 216 del Kendall

beta1=(sqrbeta1^2);
r=(6*(beta2-beta1-1))/(2*beta2-3*beta1-6);
m=(r+2)/2;
nu=-(r*(r-2)*sqrbeta1)/(sqrt(16*(r-1)-beta1*((r-2)^2)));
sigma=1; %se fija porque no queda determinado por sesgo y kurtosis
a=sqrt(((sigma^2)/16)*(16*(r-1)-beta1*((r-2)^2)));
y=k.*((1+((x./a).^2)).^(-m)).*exp(-nu*atan(x./a));
```

LISTADO 8: Código MATLAB para Distribuciones de Pearson

```
        function y=ivrnd(sqrbeta1,beta2,k,N,M)

%function y=ivrnd(sqrbeta1,beta2,k,N,M)
%
%genera muestras de la distribucion tipo IV de Pearson
%utiliza el metodo de rejection
%como curva mayorante se toma la lorentziana
%sqrbeta1: parametro de sesgo (Fisher skewness=mu3/sigma^3)
%beta2: parametro de kurtosis (Pearson kurtosis=mu4/sigma^4)
%k: constante de normalizacion para que la integral sea 1
```

```
%N: tamaño de la muestra
%M: numero de muestras
%ver pag. 216 del Kendall
%ver pag. 292 del Numerical Recipes

N1=(5*N);        %para garantizar que se generan un numero suficiente de muestras
beta1=(sqrbeta1^2);
r=(6*(beta2-beta1-1))/(2*beta2-3*beta1-6);
m=(r+2)/2;
nu=-(r*(r-2)*sqrbeta1)/(sqrt(16*(r-1)-beta1*((r-2)^2)));
sigma=1; %se fija porque no queda determinado por sesgo y kurtosis
a=sqrt(((sigma^2)/16)*(16*(r-1)-beta1*((r-2)^2)));
media=-(a*nu)/r;
x0=media;        %centramos la Lorentziana en la media de la tipo IV
a0=1;
c0=2;            %parametros de la curva mayorante g(x)
area=a0*c0;      %area de g(x), con area 2 parece ir bien
G=unifrnd(0,area,N1,M);
x1=a0*tan(((G*pi)/(a0*c0))-(pi/2))+x0;
gx1=(c0/pi)./(1+(((x1-x0).^2)/(a0^2)));
x2=unifrnd(0,gx1,N1,M);
f=k.*((1+((x1./a).^2)).^(-m)).*exp(-nu*atan(x1./a));
validos=x2<=f;
y=[];
for i=1:M
        x11=x1(:,i);
        x11=x11(validos(:,i));
        y=[y x11(1:N)];
end
```

LISTADO 9: Código MATLAB para Tests Clásicos de Normalidad

```
        function [z,p]=Dagostino_skew(muestra)

%function [z,p]=Dagostino_skew(muestra)
%
%TEST DE NORMALIDAD BASADO EN EL SESGO (D'AGOSTINO)
%z: vector con los estadisticos para el sesgo referidos a N(0,1)
%p: vector con los p-values asociados
%muestra: matriz de muestras

N=size(muestra,1);       %numero de filas (tamaño de la muestra)
M=size(muestra,2);       %numero de columnas (numero de muestras)

sesgo=skewness(muestra);
%se calcula el valor del estadistico de prueba que tiene una distribucion
%aproximadamente normal
Y=sesgo.*sqrt(((N+1).*(N+3))./(6.*(N-2)));
beta2=(3.*(N.^2+27.*N-70).*(N+1).*(N+3))./((N-2).*(N+5).*(N+7).*(N+9));
W2=-1+sqrt(2.*(beta2-1));
delta=1./sqrt(log(sqrt(W2)));
alfa=sqrt(2./(W2-1));
z=delta.*log((Y./alfa)+sqrt(((Y./alfa).^2)+1));
p=zeros(1,M);
for i=1:M
        area=normcdf(z(i));
        if area >= .5
        p(i)=1-area;
        else
                p(i)=area;
        end
end
```

LISTADO 10: Código MATLAB para Tests Clásicos de Normalidad

```
        function [z,p,indice]=Dagostino_kurto(muestra)

%function [z,p,indice]=Dagostino_kurto(muestra)
%
%TEST DE NORMALIDAD BASADO EN LA KURTOSIS (ANSCOMBE AND GLYNN)
%z: vector con los estadisticos para el sesgo referidos a N(0,1)
%p: vector con los p-values asociados
%indice: vector de indices con valores conflictivos
%muestra: matriz de muestras

N=size(muestra,1);       %numero de filas (tamaño de la muestra)
M=size(muestra,2);       %numero de columnas (numero de muestras)

kurto=kurtosis(muestra);
%se calcula el valor del estadistico de prueba que tiene una distribucion
%aproximadamente normal
Ekurto=(3.*(N-1))./(N+1);
varkurto=(24.*N.*(N-2).*(N-3))./((N+1).^2.*(N+3).*(N+5));
x=(kurto-Ekurto)./sqrt(varkurto);
beta1=((6.*(N.^2-5.*N+2))./((N+7).*(N+9))).*sqrt((6.*(N+3).*(N+5))./(N.*(N-2).*(N-3)));
A=6+((8./beta1).*(2./beta1+sqrt(1+(4./(beta1.^2)))));
z=((1-(2./(9.*A)))-((1-(2./A))./(1+(x.*sqrt(2./(A-4))))).^(1./3))./sqrt(2./(9.*A));
%size(z)
indice=z==real(z);
indice=not(indice);
z(indice)=[];
p=zeros(1,length(z));
for i=1:length(p)
        area=normcdf(z(i));
        if area >= .5
        p(i)=1-area;
        else
                p(i)=area;
        end
end
```

LISTADO 11: Código MATLAB para Tests Clásicos de Normalidad

```
        function [chi,p,muestra]=Dagostino_omni(muestra)

%function [chi,p,muestra]=Dagostino_omni(muestra)
%
%TEST DE NORMALIDAD BASADO EN EL SESGO Y LA KURTOSIS (D'AGOSTINO AND PEARSON)
%y: valor del estadistico de prueba para la muestra
%muestra: vector de muestras

N=size(muestra,1);       %numero de filas (tamaño de la muestra)
M=size(muestra,2);       %numero de columnas (numero de muestras)

%se calcula el valor del estadistico de prueba a partir de los estadisticos
%de sesgo y kurtosis por separado
z1=Dagostino_skew(muestra);
[z2,p,indice]=Dagostino_kurto(muestra);

%se elevan los estadisticos al cuadrado y se suman resultando un estadistico
%con aproximadamente una distribucion chi-cuadrado
z1(indice)=[];
muestra(:,indice)=[];
chi=z1.^2+z2.^2;
p=1-chi2cdf(chi,2);
```

LISTADO 12: Código MATLAB para Tests Clásicos de Normalidad

```
        function [potencias,rejilla]=shapiro_wilk_test(rejx,rejy,N,M,error1)

%function [potencias,rejilla]=shapiro_wilk_test(rejx,rejy,N,M,error1)
%
%test de Shapiro-Wilk entre la normal y cada pto. de la rejilla especificada
%potencias: vector conteniendo las potencias en cada pto.
%rejilla: rejilla con los ptos. validos
%rejx: vector con los puntos a tomar en el eje de sesgo (Fisher skewness=mu3/sigma^3)
%rejy: vector con los puntos a tomar en el eje de kurtosis (Pearson kurtosis=mu4/sigma^4)
%N: tamaño de muestra con la que se trabaja
%M: numero de muestras de cada pto. para hacer los tests
%error1: valor de referencia para el error tipo I

rejilla=kronb(rejx',rejy');
npuntos=size(rejilla,1);
indices=[];
potencias=[];

for i=1:npuntos
        i
        %alternativa(i,1), alternativa(i,2)
        parametros=johnson(0,1,rejilla(i,1),rejilla(i,2));
        tipo=parametros(5);
        ifault=parametros(6);
       if ifault == 1 | ifault == 2 | tipo == 5
                indices=[indices i];
                potencias=[potencias inf];
        else

        muestra=johngen(N,M,parametros(5),parametros(1),parametros(2),parametros(3),parametros(4));
                [w,z,p]=shapiro_wilk(muestra);
                indice_error=p < error1;
                potencias=[potencias sum(indice_error)/M];
        end
end
rejilla(indices,:)=[];
```

LISTADO 13: Código MATLAB para Tests Clásicos de Normalidad

```
        function [z,muestra]=zp(muestra)

%function [z,muestra]=zp(muestra)
%
%TEST DE NORMALIDAD BASADO EN LA FISHER Z-TRANSFORM
%z: vector con los estadisticos
%muestra: matriz de muestras

N=size(muestra,1);       %numero de filas (tamaño de la muestra)
M=size(muestra,2);       %numero de columnas (numero de muestras)

muestra1=muestra;
us=zeros(N,M);
r=zeros(1,M);
for i=1:N
        muestra1(i,:)=[];
       us(i,:)=(sum(muestra1.^2)-(sum(muestra1).^2) ./(N-1)).^(1/3);
        muestra1=muestra;
end
for i=1:M
        matcorr=corrcoef(muestra(:,i),us(:,i));
        r(i)=matcorr(1,2);
end
z=atanh(r);
indice1=z==real(z);
indice1=not(indice1);
indice2=isinf(z);
z(indice1 | indice2)=[];
muestra(:,indice1 | indice2)=[];
```

LISTADO 14: Código MATLAB para Tests Clásicos de Normalidad

```
        function [y]=k_mn(muestra,m)

%function [y]=k_mn(muestra,m)
%
%familia de estadisticos propuesta por Vasicek motivado por una caracterizacion de entropia maximal de la
normalidad
%y: vector con los estadisticos
%muestra: matriz de muestras
%m: parametro a fijar para calcular el estadistico

N=size(muestra,1);       %numero de filas (tamaño de la muestra)
M=size(muestra,2);       %numero de columnas (numero de muestras)

muestra=sort(muestra);
prod_dif=ones(1,M);
for i=1:N
        indice1=i+m;
        indice2=i-m;
        if i+m > N
                indice1=N;
        elseif i-m < 1
                indice2=1;
        end
        prod_dif=prod_dif.*(muestra(indice1,:)-muestra(indice2,:));
end
y=(N./(2.*m.*std(muestra))).*(prod_dif).^(1/N);
```

LISTADO 15: Código CML para el Problema del Diseño de Clasificadores

```
KNOWLEDGE-MODEL Modelo-Diseno-Clasificador;


DOMAIN-KNOWLEDGE Dominio-Diseno-Clasificador;

DOMAIN-SCHEMA Esquema-Dominio-Diseno-Clasificador;

CONCEPT Clasificador;
        DESCRIPTION: "Objeto que describe el clasificador a través de una serie de parámetros
generales";
        ATTRIBUTES:
                Numero-Entradas: NATURAL;         /*dimensionalidad*/
                Numero-Salidas: NATURAL;          /*n° de clases*/
                Priors-Clases: REAL;              /*si no se especifican, se calculan a partir de los
patrones de entrada*/
                Numero-Patrones-Entrenamiento: NATURAL;   /*en cada clase*/
                Numero-Patrones-Validacion: NATURAL;      /*en cada clase*/
                Estimacion-Error: STRING;         /*método de Estimacion del error*/
                Error-Clasificacion: REAL;                /*Estimacion del error*/
                Indice-Ordenacion: NATURAL;               /*índice de ordenación*/
END CONCEPT Clasificador;

CONCEPT Parametrico;
        DESCRIPTION: "Clase de clasificadores paramétricos";
        SUB-TYPE-OF: Clasificador;
END CONCEPT Parametrico;

CONCEPT No-Parametrico;
        DESCRIPTION: "Clase de clasificadores no paramétricos";
        SUB-TYPE-OF: Clasificador;
END CONCEPT No-Parametrico;
```

```
CONCEPT Otros;
        DESCRIPTION: "Otro tipo de clasificadores";
        SUB-TYPE-OF: Clasificador;
END CONCEPT Otros;

CONCEPT Lineal;
        DESCRIPTION: "Clase de discriminantes lineales";
        SUB-TYPE-OF: Parametrico;
        ATTRIBUTES:
                Tipo-Discriminante: STRING;              /*tipo de discriminante lineal*/
                Estimacion-Parametros: STRING;    /*método de Estimacion de parámetros*/
END CONCEPT Lineal;

CONCEPT Cuadratico;
        DESCRIPTION: "Clase de discriminantes cuadráticos";
        SUB-TYPE-OF: Parametrico;
        ATTRIBUTES:
                Estimacion-Parametros: STRING;    /*método de Estimacion de parámetros*/
END CONCEPT Cuadratico;

CONCEPT Flexible;
        DESCRIPTION: "Clase de discriminantes más flexibles que los lineales o cuadráticos";
        SUB-TYPE-OF: Parametrico;
END CONCEPT Flexible;

CONCEPT Red-Neuronal;
        DESCRIPTION: "Clase de las redes neuronales";
        SUB-TYPE-OF: Flexible;
        ATTRIBUTES:
                Numero-Nodos: NATURAL;           /*nodos en la capa oculta*/
                Algoritmo-Entrenamiento: STRING;
                Weight-Decay: REAL;      /*parametro de weight-decay*/
END CONCEPT Red-Neuronal;

CONCEPT Vecinos-Proximos;
        DESCRIPTION: "tipo de discriminante no parametrico";
        SUB-TYPE-OF: No-Parametrico;
        ATTRIBUTES:
                Numero-Vecinos: NATURAL; /*Numero de vecinos a escogerl*/
                Metrica: STRING; /*métrica utilizada*/
END CONCEPT Vecinos-Proximos;

CONCEPT Patrones-Entrada;
        DESCRIPTION: "Objeto que apunta al conjunto de patrones etiquetados de entrada";
        ATTRIBUTES:
                Identificador-Patrones: STRING;
END CONCEPT Patrones-Entrada;

CONCEPT Informacion-Patrones;
        DESCRIPTION: "Informacion extraída de los patrones de entrada";
        ATTRIBUTES:
                Numero-Patrones-Entrada: NATURAL; /*en cada clase*/
                Numero-Clases: NATURAL;          /*n° de clases*/
                Numero-Caracteristicas: NATURAL;          /*n° de caracteristicas*/
                Test1-Normalidad-Mono: REAL;              /*probabilidades posteriori en cada clase*/
                Test2-Normalidad-Mono: REAL;              /*probabilidades posteriori en cada clase*/
                Test1-Normalidad-Multi: REAL;     /*valores P en cada clase*/
                Distancia-Bhattacharyya: REAL;
                Distancia-Bhattacharyya-1: REAL;  /*termino lineal*/
                Distancia-Bhattacharyya-2: REAL;  /*termino cuadrático*/
                Sesgo-Lineal: REAL;
                Sesgo-Cuadratico: REAL;
                Dimensionalidad-Intrinseca: REAL;
                Reduccion-Dimensionalidad: STRING;                /*metodo reduccion dimensionalidad*/
END CONCEPT Informacion-Patrones;

CONCEPT Prestaciones-Clasificador;
        DESCRIPTION: "Prestaciones a alcanzar por el clasificador";
        ATTRIBUTES:
                Error-Promedio-Deseado: REAL;
END CONCEPT Prestaciones-Clasificador;

CONCEPT Violaciones-Clasificador;
        DESCRIPTION: "Prestaciones no alcanzadas por el clasificador";
        ATTRIBUTES:
                Error-Promedio-Alcanzado: STRING; /*SI-NO*/
END CONCEPT Violaciones-Clasificador;

CONCEPT Clasificadores-Validos;
        DESCRIPTION: "Conjunto de clasificadores validos";
```

```
END CONCEPT Clasificadores-Validos;

CONCEPT Lista-Clasificadores;
        DESCRIPTION: "Lista ordenada de clasificadores validos";
END CONCEPT Lista-Clasificadores;

RULE-TYPE Dependencia-Clasificador-Patrones-1;
        DESCRIPTION: "Tipo de Regla para proponer un modelo de clasificador";
        ANTECEDENT: Informacion-Patrones, Lineal;
        CONSEQUENT: Lineal;
        CONNECTION-SYMBOL: sugiere-1;
END RULE-TYPE Dependencia-Clasificador-Patrones-1;

RULE-TYPE Dependencia-Clasificador-Patrones-2;
        DESCRIPTION: "Tipo de Regla para proponer un modelo de clasificador";
        ANTECEDENT: Informacion-Patrones, Cuadratico;
        CONSEQUENT: Cuadratico;
        CONNECTION-SYMBOL: sugiere-2;
END RULE-TYPE Dependencia-Clasificador-Patrones-2;

RULE-TYPE Dependencia-Clasificador-Patrones-3;
        DESCRIPTION: "Tipo de Regla para proponer un modelo de clasificador";
        ANTECEDENT: Informacion-Patrones, Red-Neuronal;
        CONSEQUENT: Red-Neuronal;
        CONNECTION-SYMBOL: sugiere-3;
END RULE-TYPE Dependencia-Clasificador-Patrones-3;

RULE-TYPE Dependencia-Clasificador-Patrones-4;
        DESCRIPTION: "Tipo de Regla para proponer un modelo de clasificador";
        ANTECEDENT: Informacion-Patrones, Vecinos-Proximos;
        CONSEQUENT: Vecinos-Proximos;
        CONNECTION-SYMBOL: sugiere-4;
END RULE-TYPE Dependencia-Clasificador-Patrones-4;

RULE-TYPE Relacion-Entre-Clasificadores-1;
        DESCRIPTION: "Tipo de Regla para criticar un modelo de clasificador";
        ANTECEDENT: Violaciones-Clasificador, Lineal;
        CONSEQUENT: Lineal;
        CONNECTION-SYMBOL: cambiar-a-1;
END RULE-TYPE Relacion-Entre-Clasificadores-1;

RULE-TYPE Relacion-Entre-Clasificadores-2;
        DESCRIPTION: "Tipo de Regla para criticar un modelo de clasificador";
        ANTECEDENT: Violaciones-Clasificador, Cuadratico;
        CONSEQUENT: Cuadratico;
        CONNECTION-SYMBOL: cambiar-a-2;
END RULE-TYPE Relacion-Entre-Clasificadores-2;

RULE-TYPE Relacion-Entre-Clasificadores-3;
        DESCRIPTION: "Tipo de Regla para criticar un modelo de clasificador";
        ANTECEDENT: Violaciones-Clasificador, Red-Neuronal;
        CONSEQUENT: Red-Neuronal;
        CONNECTION-SYMBOL: cambiar-a-3;
END RULE-TYPE Relacion-Entre-Clasificadores-3;

RULE-TYPE Relacion-Entre-Clasificadores-4;
        DESCRIPTION: "Tipo de Regla para criticar un modelo de clasificador";
        ANTECEDENT: Violaciones-Clasificador, Vecinos-Proximos;
        CONSEQUENT: Vecinos-Proximos;
        CONNECTION-SYMBOL: cambiar-a-4;
END RULE-TYPE Relacion-Entre-Clasificadores-4;

RULE-TYPE Relacion-Entre-Patrones;
        DESCRIPTION: "Tipo de Regla para criticar los patrones de entrada";
        ANTECEDENT: Informacion-Patrones;
        CONSEQUENT: Informacion-Patrones;
        CONNECTION-SYMBOL: cambiar-a;
END RULE-TYPE Relacion-Entre-Patrones;

RULE-TYPE Relacion-Ordenacion-1;
        DESCRIPTION: "Tipo de Regla para ordenar los clasificadores válidos";
        ANTECEDENT: Informacion-Patrones, Lineal;
        CONSEQUENT: Lineal;
        CONNECTION-SYMBOL: ordenar-como-1;
END RULE-TYPE Relacion-Ordenacion-1;

RULE-TYPE Relacion-Ordenacion-2;
        DESCRIPTION: "Tipo de Regla para ordenar los clasificadores válidos";
        ANTECEDENT: Informacion-Patrones, Cuadratico;
        CONSEQUENT: Cuadratico;
```

```
        CONNECTION-SYMBOL: ordenar-como-2;
END RULE-TYPE Relacion-Ordenacion-2;

RULE-TYPE Relacion-Ordenacion-3;
        DESCRIPTION: "Tipo de Regla para ordenar los clasificadores válidos";
        ANTECEDENT: Informacion-Patrones, Red-Neuronal;
        CONSEQUENT: Red-Neuronal;
        CONNECTION-SYMBOL: ordenar-como-3;
END RULE-TYPE Relacion-Ordenacion-3;

RULE-TYPE Relacion-Ordenacion-4;
        DESCRIPTION: "Tipo de Regla para ordenar los clasificadores válidos";
        ANTECEDENT: Informacion-Patrones, Vecinos-Proximos;
        CONSEQUENT: Vecinos-Proximos;
        CONNECTION-SYMBOL: ordenar-como-4;
END RULE-TYPE Relacion-Ordenacion-4;

END DOMAIN-SCHEMA Esquema-Dominio-Diseno-Clasificador;

KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-1;
        USES: Dependencia-Clasificador-Patrones-1 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2 AND Informacion-Patrones.Distancia-
Bhattacharyya-1 > 0.25 AND Informacion-Patrones.Numero-Patrones-Entrada > 10*Informacion-Patrones.Sesgo-
Lineal
                == sugiere-1 =>
                Lineal.Tipo-Discriminante = Fisher AND Lineal.Estimacion-Parametros = ML AND
Lineal.Estimacion-Error = Leave-One-Out;
END KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-1;

KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-2;
        USES: Dependencia-Clasificador-Patrones-2 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2 AND Informacion-Patrones.Distancia-
Bhattacharyya-2 > 0.25 AND Informacion-Patrones.Numero-Patrones-Entrada > 10*Informacion-Patrones.Sesgo-
Cuadratico
                == sugiere-2 =>
                Lineal.Estimacion-Parametros = ML AND Lineal.Estimacion-Error = Leave-One-Out;
END KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-2;

KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-3;
        USES: Dependencia-Clasificador-Patrones-3 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2
                == sugiere-3 =>
                Red-Neuronal.Numero-Nodos = Informacion-Patrones.Numero-Caracteristicas AND Red-
Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-Neuronal.Weight-Decay = 0 AND Red-
Neuronal.Estimacion-Error = Leave-One-Out;
                Informacion-Patrones.Numero-Clases >= 2
                == sugiere-3 =>
                Red-Neuronal.Numero-Nodos = 2*Informacion-Patrones.Numero-Caracteristicas AND Red-
Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-Neuronal.Weight-Decay = 0 AND Red-
Neuronal.Estimacion-Error = Leave-One-Out;
                Informacion-Patrones.Numero-Clases >= 2
                == sugiere-3 =>
                Red-Neuronal.Numero-Nodos = 3*Informacion-Patrones.Numero-Caracteristicas AND Red-
Neuronal.Algoritmo-Entrenamiento = Quasi-Newton AND Red-Neuronal.Weight-Decay = 0 AND Red-
Neuronal.Estimacion-Error = Leave-One-Out;
END KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-3;

KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-4;
        USES: Dependencia-Clasificador-Patrones-4 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2
                == sugiere-4 =>
                Vecinos-Proximos.Numero-Vecinos = 1 AND Vecinos-Proximos.Metrica = Euclidea AND Vecinos-
Proximos.Estimacion-Error = Leave-One-Out;
END KNOWLEDGE-BASE Propuestas-Modelos-Clasificadores-4;

KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-1;
        USES: Relacion-Entre-Clasificadores-1 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Lineal.Estimacion-Parametros = ML
                == cambiar-a-1 =>
                Lineal.Estimacion-Parametros = t-Multivariable;
                Lineal.Estimacion-Parametros = ML
                == cambiar-a-1 =>
                Lineal.Estimacion-Parametros = MVE;
                Lineal.Estimacion-Parametros = ML
                == cambiar-a-1 =>
```

```
                Lineal.Estimacion-Parametros = Debiased;
                Lineal.Estimacion-Parametros = ML
                == cambiar-a-1 =>
                Lineal.Estimacion-Parametros = Predictivo;
END KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-1;

KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-2;
        USES: Relacion-Entre-Clasificadores-2 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Cuadratico.Estimacion-Parametros = ML
                == cambiar-a-2 =>
                Cuadratico.Estimacion-Parametros = t-Multivariable;
                Cuadratico.Estimacion-Parametros = ML
                == cambiar-a-2 =>
                Cuadratico.Estimacion-Parametros = MVE;
                Cuadratico.Estimacion-Parametros = ML
                == cambiar-a-2 =>
                Cuadratico.Estimacion-Parametros = Debiased;
                Cuadratico.Estimacion-Parametros = ML
                == cambiar-a-2 =>
                Cuadratico.Estimacion-Parametros = Predictivo;
END KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-2;

KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-3;
        USES: Relacion-Entre-Clasificadores-3 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Red-Neuronal.Weight-Decay = 0
                == cambiar-a-3 =>
                Red-Neuronal.Weight-Decay = 0.001;
                Red-Neuronal.Weight-Decay = 0
                == cambiar-a-3 =>
                Red-Neuronal.Weight-Decay = 0.01;
                Red-Neuronal.Weight-Decay = 0
                == cambiar-a-3 =>
                Red-Neuronal.Weight-Decay = 0.1;
END KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-3;

KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-4;
        USES: Relacion-Entre-Clasificadores-4 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Vecinos-Proximos.Numero-Vecinos = 1
                == cambiar-a-4 =>
                Vecinos-Proximos.Numero-Vecinos = 2;
                Vecinos-Proximos.Numero-Vecinos = 1
                == cambiar-a-4 =>
                Vecinos-Proximos.Numero-Vecinos = 3;
END KNOWLEDGE-BASE Criticas-Modelos-Clasificadores-4;

KNOWLEDGE-BASE Criticas-Patrones-Entrada;
        USES: Relacion-Entre-Patrones FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Reduccion-Dimensionalidad = ninguno
                == cambiar-a =>
                Informacion-Patrones.Reduccion-Dimensionalidad = Componentes-Principales;
                Informacion-Patrones.Reduccion-Dimensionalidad = ninguno
                == cambiar-a =>
                Informacion-Patrones.Reduccion-Dimensionalidad = Fisher;
END KNOWLEDGE-BASE Criticas-Patrones-Entrada;

KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-1;
        USES: Relacion-Ordenacion-1 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Patrones-Entrada >= 100 AND (Informacion-Patrones.Test1-
Normalidad-Mono > 0.75 OR Informacion-Patrones.Test2-Normalidad-Mono > 0.75) AND Informacion-
Patrones.Test1-Normalidad-Multi > 3.5
                == ordenar-como-1 =>
                Lineal.Indice-Ordenacion = 1;
                Informacion-Patrones.Numero-Clases >= 2
                == ordenar-como-1 =>
                Lineal.Indice-Ordenacion = 1;
END KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-1;

KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-2;
        USES: Relacion-Ordenacion-2 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Patrones-Entrada >= 100 AND (Informacion-Patrones.Test1-
Normalidad-Mono > 0.75 OR Informacion-Patrones.Test2-Normalidad-Mono > 0.75) AND Informacion-
Patrones.Test1-Normalidad-Multi > 3.5
                == ordenar-como-2 =>
                Cuadratico.Indice-Ordenacion = 2;
```

```
                Informacion-Patrones.Numero-Clases >= 2
                == ordenar-como-2 =>
                Cuadratico.Indice-Ordenacion = 2;
END KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-2;

KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-3;
        USES: Relacion-Ordenacion-3 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2
                == ordenar-como-3 =>
                Red-Neuronal.Indice-Ordenacion = 3;
END KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-3;

KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-4;
        USES: Relacion-Ordenacion-4 FROM Esquema-Dominio-Diseno-Clasificador;
        EXPRESSIONS:
                Informacion-Patrones.Numero-Clases >= 2
                == ordenar-como-4 =>
                Vecinos-Proximos.Indice-Ordenacion = 4;
END KNOWLEDGE-BASE Criterios-Ordenacion-Clasificadores-4;

END DOMAIN-KNOWLEDGE Dominio-Diseno-Clasificador;

INFERENCE-KNOWLEDGE Inferencias-Diseno-Clasificador;

KNOWLEDGE-ROLE Restricciones-Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Patrones-Entrada;
END KNOWLEDGE-ROLE Restricciones-Diseno;

KNOWLEDGE-ROLE Restricciones-Explicitas;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Informacion-Patrones;
END KNOWLEDGE-ROLE Restricciones-Explicitas;

KNOWLEDGE-ROLE Estructura-Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Clasificador;
END KNOWLEDGE-ROLE Estructura-Diseno;

KNOWLEDGE-ROLE Propuesta-Diseno-1;
 TYPE: STATIC;
 DOMAIN-MAPPING: Dependencia-Clasificador-Patrones-1 FROM Propuestas-Modelos-Clasificadores-1;
END KNOWLEDGE-ROLE Propuesta-Diseno-1;

KNOWLEDGE-ROLE Propuesta-Diseno-2;
 TYPE: STATIC;
 DOMAIN-MAPPING: Dependencia-Clasificador-Patrones-2 FROM Propuestas-Modelos-Clasificadores-2;
END KNOWLEDGE-ROLE Propuesta-Diseno-2;

KNOWLEDGE-ROLE Propuesta-Diseno-3;
 TYPE: STATIC;
 DOMAIN-MAPPING: Dependencia-Clasificador-Patrones-3 FROM Propuestas-Modelos-Clasificadores-3;
END KNOWLEDGE-ROLE Propuesta-Diseno-3;

KNOWLEDGE-ROLE Propuesta-Diseno-4;
 TYPE: STATIC;
 DOMAIN-MAPPING: Dependencia-Clasificador-Patrones-4 FROM Propuestas-Modelos-Clasificadores-4;
END KNOWLEDGE-ROLE Propuesta-Diseno-4;

KNOWLEDGE-ROLE Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Clasificador;
END KNOWLEDGE-ROLE Diseno;

KNOWLEDGE-ROLE Prestaciones-Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Prestaciones-Clasificador;
END KNOWLEDGE-ROLE Prestaciones-Diseno;

KNOWLEDGE-ROLE Violaciones-Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Violaciones-Clasificador;
END KNOWLEDGE-ROLE Violaciones-Diseno;

KNOWLEDGE-ROLE Valor-Verdad;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: TRUTH-VALUE;
END KNOWLEDGE-ROLE Valor-Verdad;
```

```
KNOWLEDGE-ROLE Disenos-Validos;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: LIST;
END KNOWLEDGE-ROLE Disenos-Validos;

KNOWLEDGE-ROLE Critica-Diseno-1;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Entre-Clasificadores-1 FROM Criticas-Modelos-Clasificadores-1;
END KNOWLEDGE-ROLE Critica-Diseno-1;

KNOWLEDGE-ROLE Critica-Diseno-2;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Entre-Clasificadores-2 FROM Criticas-Modelos-Clasificadores-2;
END KNOWLEDGE-ROLE Critica-Diseno-2;

KNOWLEDGE-ROLE Critica-Diseno-3;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Entre-Clasificadores-3 FROM Criticas-Modelos-Clasificadores-3;
END KNOWLEDGE-ROLE Critica-Diseno-3;

KNOWLEDGE-ROLE Critica-Diseno-4;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Entre-Clasificadores-4 FROM Criticas-Modelos-Clasificadores-4;
END KNOWLEDGE-ROLE Critica-Diseno-4;

KNOWLEDGE-ROLE Modificaciones-Diseno;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Clasificador;
END KNOWLEDGE-ROLE Modificaciones-Diseno;

KNOWLEDGE-ROLE Modificaciones-Restricciones;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: Informacion-Patrones;
END KNOWLEDGE-ROLE Modificaciones-Restricciones;

KNOWLEDGE-ROLE Critica-Restricciones;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Entre-Patrones FROM Criticas-Patrones-Entrada;
END KNOWLEDGE-ROLE Critica-Restricciones;

KNOWLEDGE-ROLE Criterio-Ordenacion-1;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Ordenacion-1 FROM Criterios-Ordenacion-Clasificadores-1;
END KNOWLEDGE-ROLE Criterio-Ordenacion-1;

KNOWLEDGE-ROLE Criterio-Ordenacion-2;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Ordenacion-2 FROM Criterios-Ordenacion-Clasificadores-2;
END KNOWLEDGE-ROLE Criterio-Ordenacion-2;

KNOWLEDGE-ROLE Criterio-Ordenacion-3;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Ordenacion-3 FROM Criterios-Ordenacion-Clasificadores-3;
END KNOWLEDGE-ROLE Criterio-Ordenacion-3;

KNOWLEDGE-ROLE Criterio-Ordenacion-4;
 TYPE: STATIC;
 DOMAIN-MAPPING: Relacion-Ordenacion-4 FROM Criterios-Ordenacion-Clasificadores-4;
END KNOWLEDGE-ROLE Criterio-Ordenacion-4;

KNOWLEDGE-ROLE Lista-Disenos;
 TYPE: DYNAMIC;
 DOMAIN-MAPPING: LIST;
END KNOWLEDGE-ROLE Lista-Disenos;

INFERENCE Explicitar-Restricciones;
 ROLES:
  INPUT: Restricciones-Diseno;
  OUTPUT: Restricciones-Explicitas;
 SPECIFICATION: "Hace explícitas las restricciones";
END INFERENCE Explicitar-Restricciones;

INFERENCE Especificar-Estructura-Diseno;
 ROLES:
  INPUT: Restricciones-Explicitas;
  OUTPUT: Estructura-Diseno;
  STATIC: Propuesta-Diseno-1, Propuesta-Diseno-2, Propuesta-Diseno-3, Propuesta-Diseno-4;
 SPECIFICATION: "A partir de los requerimientos explícitos, especifica una estructura de Diseno";
END INFERENCE Especificar-Estructura-Diseno;
```

```
INFERENCE Proponer-Diseno;
 ROLES:
  INPUT: Restricciones-Diseno, Estructura-Diseno;
  OUTPUT: Diseno;
 SPECIFICATION: "Propone un Diseno";
END INFERENCE Proponer-Diseno;

INFERENCE Verificar-Diseno;
 ROLES:
  INPUT: Diseno, Prestaciones-Diseno;
  OUTPUT: Disenos-Validos, Violaciones-Diseno, Valor-Verdad;
 SPECIFICATION: "Verifica el Diseno";
END INFERENCE Verificar-Diseno;

INFERENCE Criticar-Diseno;
 ROLES:
  INPUT: Diseno, Violaciones-Diseno;
  OUTPUT: Modificaciones-Diseno;
  STATIC: Critica-Diseno-1, Critica-Diseno-2, Critica-Diseno-3, Critica-Diseno-4;
  SPECIFICATION: "Critica el Diseno proponiendo modificaciones";
END INFERENCE Criticar-Diseno;

INFERENCE Modificar-Diseno;
 ROLES:
  INPUT: Modificaciones-Diseno;
  OUTPUT: Diseno;
 SPECIFICATION: "Modifica el Diseno";
END INFERENCE Modificar-Diseno;

INFERENCE Criticar-Restricciones;
 ROLES:
  INPUT: Restricciones-Explicitas;
  OUTPUT: Modificaciones-Restricciones;
  STATIC: Critica-Restricciones;
  SPECIFICATION: "Critica las restricciones proponiendo modificaciones";
END INFERENCE Criticar-Restricciones;

INFERENCE Modificar-Restricciones;
 ROLES:
  INPUT: Restricciones-Diseno, Modificaciones-Restricciones;
  OUTPUT: Restricciones-Diseno;
 SPECIFICATION: "Modifica las restricciones";
END INFERENCE Modificar-Restricciones;

INFERENCE Ordenar-Disenos;
 ROLES:
  INPUT: Restricciones-Explicitas, Disenos-Validos;
  OUTPUT: Lista-Disenos;
  STATIC: Criterio-Ordenacion-1, Criterio-Ordenacion-2, Criterio-Ordenacion-3, Criterio-Ordenacion-4;
  SPECIFICATION: "Critica las restricciones proponiendo modificaciones";
END INFERENCE Ordenar-Disenos;

END INFERENCE-KNOWLEDGE Inferencias-Diseno-Clasificador;

TASK-KNOWLEDGE Tareas-Diseno-Clasificador;

TASK Diseno-Clasificador;
 GOAL: "Diseñar un clasificador";
 ROLES:
  INPUT:
   Restricciones-Diseno: "Restricciones del problema de Diseno";
   Prestaciones-Diseno: "Prestaciones del problema de Diseno";
  OUTPUT:
   Lista-Disenos: "Lista de Disenos que satisfacen los requerimientos y alcanzan las prestaciones";
  METHOD: Diseno-Usando-Proponer-Criticar-Modificar;
END TASK Diseno-Clasificador;

TASK-METHOD Diseno-Usando-Proponer-Criticar-Modificar;
        REALIZES: Diseno-Clasificador;
        DECOMPOSITION:
                INFERENCES: Explicitar-Restricciones, Especificar-Estructura-Diseno, Proponer-Diseno,
Verificar-Diseno, Criticar-Diseno, Modificar-Diseno, Criticar-Restricciones, Modificar-Restricciones,
Ordenar-Disenos;
                ROLES:
                        INTERMEDIATE:
                        Restricciones-Explicitas: "Restricciones explicitas del diseno";
                        Estructura-Diseno: "Estructura del diseno";
                        Diseno: "Diseno";
                        Valor-Verdad: "Valor-Verdad";
                        Prestaciones-Diseno: "Prestaciones del diseno";
```

```
                        Violaciones-Diseno: "Violaciones del diseno";
                        Modificaciones-Diseno: "Modificaciones del diseno";
                        Modificaciones-Restricciones: "Modificaciones de las restricciones";
                        Disenos-Validos: "Conjunto de disenos validos";
                        Lista-Disenos: "Lista ordenada de disenos validos";
        CODE:
                {
                Explicitar-Restricciones(Restricciones-Diseno -> Restricciones-Explícitas);
                WHILE NEW-SOLUTION Especificar-Estructura-Diseno(Restricciones-Explícitas -> Estructura-
Diseno) DO
                Proponer-Diseno(Estructura-Diseno + Restricciones-Diseno -> Diseno);
                Verificar-Diseno(Diseno + Prestaciones-Diseno -> Violaciones-Diseno + Disenos-Válidos +
Valor-Verdad);
                WHILE Valor-Verdad == FALSE AND NEW-SOLUTION Criticar-Diseno(Diseno + Violaciones-Diseno
-> Modificaciones-Diseno) DO
                Modificar-Diseno(Modificaciones-Diseno -> Diseno);
                Verificar-Diseno(Diseno + Prestaciones-Diseno -> Violaciones-Diseno + Disenos-Válidos +
Valor-Verdad);
                END-WHILE
                END WHILE
                IF EMPTY Disenos-Válidos THEN
                WHILE NEW-SOLUTION Criticar-Restricciones(Restricciones-Explícitas -> Modificaciones-
Restricciones) DO
                Modificar-Restricciones(Restricciones-Diseno + Modificaciones-Restricciones ->
Restricciones-Diseno);
                Explicitar-Restricciones(Restricciones-Diseno -> Restricciones-Explícitas);
                WHILE NEW-SOLUTION Especificar-Estructura-Diseno(Restricciones-Explícitas -> Estructura-
Diseno) DO
                Proponer-Diseno(Estructura-Diseno + Restricciones-Diseno -> Diseno);
                Verificar-Diseno(Diseno + Prestaciones-Diseno -> Violaciones-Diseno + Disenos-Válidos +
Valor-Verdad);
                WHILE Valor-Verdad == FALSE AND NEW-SOLUTION Criticar-Diseno(Diseno+Violaciones-Diseno -
> Modificaciones-Diseno) DO
                Modificar-Diseno(Modificaciones-Diseno -> Diseno);
                Verificar-Diseno(Diseno + Prestaciones-Diseno -> Violaciones-Diseno + Disenos-Válidos +
Valor-Verdad);
                END-WHILE
                END WHILE
                END WHILE
                END IF
                Ordenar-Disenos(Disenos-Validos -> Lista-Disenos);
                };
END TASK-METHOD Diseno-Usando-Proponer-Criticar-Modificar;

END TASK-KNOWLEDGE Tareas-Diseno-Clasificador;


END KNOWLEDGE-MODEL Modelo-Diseno-Clasificador;
```

LISTADO 16: Código CLIPS para el Problema del Diseño de Clasificadores

```
;;;------------------------------------------------------

;;;     inicializacion


(clear)
(reset)
(defmodule MAIN (export ?ALL))

(batch* "/usr/lib/scml-lib/list.bat")
(batch* "/usr/lib/scml-lib/commonkads.clp")
(batch* "/usr/lib/scml-lib/UF-general.bat")
;;;------------------------------------------------------

;;;     DOMAIN-KNOWLEDGE general-domain-knowledge


;;;     domain schemas:

(batch* "DS-general-domain-schema.bat")
```

```
;;;     knowledge-bases:

(batch* "KB-KB-general.bat")
;;;------------------------------------------------------

;;;     DOMAIN-KNOWLEDGE Dominio-Diseno-Clasificador


;;;     domain schemas:

(batch* "DS-Esquema-Dominio-Diseno-Clasificador.bat")


;;;     knowledge-bases:

(batch* "KB-Propuestas-Modelos-Clasificadores-1.bat")
(batch* "KB-Propuestas-Modelos-Clasificadores-2.bat")
(batch* "KB-Propuestas-Modelos-Clasificadores-3.bat")
(batch* "KB-Propuestas-Modelos-Clasificadores-4.bat")
(batch* "KB-Criticas-Modelos-Clasificadores-1.bat")
(batch* "KB-Criticas-Modelos-Clasificadores-2.bat")
(batch* "KB-Criticas-Modelos-Clasificadores-3.bat")
(batch* "KB-Criticas-Modelos-Clasificadores-4.bat")
(batch* "KB-Criticas-Patrones-Entrada.bat")
(batch* "KB-Criterios-Ordenacion-Clasificadores-1.bat")
(batch* "KB-Criterios-Ordenacion-Clasificadores-2.bat")
(batch* "KB-Criterios-Ordenacion-Clasificadores-3.bat")
(batch* "KB-Criterios-Ordenacion-Clasificadores-4.bat")

;;;------------------------------------------------------

;;;     inference-knowledges:

(batch* "IK-Inferencias-Diseno-Clasificador.bat")

;;;------------------------------------------------------

;;;     task-knowledges:

(batch* "TK-Tareas-Diseno-Clasificador.bat")
;;;------------------------------------------------------

;;;     user functions :

(batch* "UF-classifier_design_2..bat")
```

LISTADO 17: Código CLIPS para el Problema del Diseño de Clasificadores

```
;;; Created with scml v1.0

(defclass TRUTH-VALUE (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "true or false concept"))
        (slot value
                        (create-accessor read-write))
)

(defclass LIST (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "ordered collection of items"))
        (multislot items
                        (create-accessor read-write))
)

(defclass NODE (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "item for the LISTs"))
        (slot key
                        (create-accessor read-write))
```

```
)
```

LISTADO 18: Código CLIPS para el Problema del Diseño de Clasificadores

```
;;; Created with scml v1.0

(defclass Clasificador (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Objeto que describe el clasificador a través de una serie de parámetros
generales"))
        (slot Numero-Entradas
                        (create-accessor read-write))
        (slot Numero-Salidas
                        (create-accessor read-write))
        (slot Priors-Clases
                        (create-accessor read-write))
        (slot Numero-Patrones-Entrenamiento
                        (create-accessor read-write))
        (slot Numero-Patrones-Validacion
                        (create-accessor read-write))
        (slot Estimacion-Error
                        (create-accessor read-write))
        (slot Error-Clasificacion
                        (create-accessor read-write))
        (slot Indice-Ordenacion
                        (create-accessor read-write))
)

(defclass Parametrico (is-a Clasificador) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de clasificadores paramétricos"))
)

(defclass No-Parametrico (is-a Clasificador) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de clasificadores no paramétricos"))
)

(defclass Otros (is-a Clasificador) (role concrete) (pattern-match reactive)
        (slot description
                (default "Otro tipo de clasificadores"))
)

(defclass Lineal (is-a Parametrico) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de discriminantes lineales"))
        (slot Tipo-Discriminante
                        (create-accessor read-write))
        (slot Estimacion-Parametros
                        (create-accessor read-write))
)

(defclass Cuadratico (is-a Parametrico) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de discriminantes cuadráticos"))
        (slot Estimacion-Parametros
                        (create-accessor read-write))
)

(defclass Flexible (is-a Parametrico) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de discriminantes más flexibles que los lineales o cuadráticos"))
)

(defclass Red-Neuronal (is-a Flexible) (role concrete) (pattern-match reactive)
        (slot description
                (default "Clase de las redes neuronales"))
        (slot Numero-Nodos
                        (create-accessor read-write))
        (slot Algoritmo-Entrenamiento
                        (create-accessor read-write))
        (slot Weight-Decay
```

```
                        (create-accessor read-write))
)

(defclass Vecinos-Proximos (is-a No-Parametrico) (role concrete) (pattern-match reactive)
        (slot description
                (default "tipo de discriminante no parametrico"))
        (slot Numero-Vecinos
                        (create-accessor read-write))
        (slot Metrica
                        (create-accessor read-write))
)

(defclass Patrones-Entrada (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Objeto que apunta al conjunto de patrones etiquetados de entrada"))
        (slot Identificador-Patrones
                        (create-accessor read-write))
)

(defclass Informacion-Patrones (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Informacion extraída de los patrones de entrada"))
        (slot Numero-Patrones-Entrada
                        (create-accessor read-write))
        (slot Numero-Clases
                        (create-accessor read-write))
        (slot Numero-Caracteristicas
                        (create-accessor read-write))
        (slot Test1-Normalidad-Mono
                        (create-accessor read-write))
        (slot Test2-Normalidad-Mono
                        (create-accessor read-write))
        (slot Test1-Normalidad-Multi
                        (create-accessor read-write))
        (slot Distancia-Bhattacharyya
                        (create-accessor read-write))
        (slot Distancia-Bhattacharyya-1
                        (create-accessor read-write))
        (slot Distancia-Bhattacharyya-2
                        (create-accessor read-write))
        (slot Sesgo-Lineal
                        (create-accessor read-write))
        (slot Sesgo-Cuadratico
                        (create-accessor read-write))
        (slot Dimensionalidad-Intrinseca
                        (create-accessor read-write))
        (slot Reduccion-Dimensionalidad
                        (create-accessor read-write))
)

(defclass Prestaciones-Clasificador (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Prestaciones a alcanzar por el clasificador"))
        (slot Error-Promedio-Deseado
                        (create-accessor read-write))
)

(defclass Violaciones-Clasificador (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Prestaciones no alcanzadas por el clasificador"))
        (slot Error-Promedio-Alcanzado
                        (create-accessor read-write))
)

(defclass Clasificadores-Validos (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Conjunto de clasificadores validos"))
)

(defclass Lista-Clasificadores (is-a CONCEPT) (role concrete) (pattern-match reactive)
        (slot description
                (default "Lista ordenada de clasificadores validos"))
)

(make-instance Dependencia-Clasificador-Patrones-1-antec of ARGUMENT
        (binding-name Informacion-Patrones Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
        )
```

```
(make-instance Dependencia-Clasificador-Patrones-1-consec of ARGUMENT
        (binding-name Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-1 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para proponer un modelo de clasificador")
        (antecedent Dependencia-Clasificador-Patrones-1-antec)
        (consequent Dependencia-Clasificador-Patrones-1-conseq)
        (connection-symbol sugiere-1)
)

(make-instance Dependencia-Clasificador-Patrones-2-antec of ARGUMENT
        (binding-name Informacion-Patrones Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-2-consec of ARGUMENT
        (binding-name Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-2 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para proponer un modelo de clasificador")
        (antecedent Dependencia-Clasificador-Patrones-2-antec)
        (consequent Dependencia-Clasificador-Patrones-2-conseq)
        (connection-symbol sugiere-2)
)

(make-instance Dependencia-Clasificador-Patrones-3-antec of ARGUMENT
        (binding-name Informacion-Patrones Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-3-consec of ARGUMENT
        (binding-name Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-3 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para proponer un modelo de clasificador")
        (antecedent Dependencia-Clasificador-Patrones-3-antec)
        (consequent Dependencia-Clasificador-Patrones-3-conseq)
        (connection-symbol sugiere-3)
)

(make-instance Dependencia-Clasificador-Patrones-4-antec of ARGUMENT
        (binding-name Informacion-Patrones Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-4-consec of ARGUMENT
        (binding-name Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Dependencia-Clasificador-Patrones-4 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para proponer un modelo de clasificador")
        (antecedent Dependencia-Clasificador-Patrones-4-antec)
        (consequent Dependencia-Clasificador-Patrones-4-conseq)
        (connection-symbol sugiere-4)
)

(make-instance Relacion-Entre-Clasificadores-1-antec of ARGUMENT
        (binding-name Violaciones-Clasificador Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-1-consec of ARGUMENT
        (binding-name Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
```

```
        )

(make-instance Relacion-Entre-Clasificadores-1 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para criticar un modelo de clasificador")
        (antecedent Relacion-Entre-Clasificadores-1-antec)
        (consequent Relacion-Entre-Clasificadores-1-conseq)
        (connection-symbol cambiar-a-1)
)

(make-instance Relacion-Entre-Clasificadores-2-antec of ARGUMENT
        (binding-name Violaciones-Clasificador Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-2-consec of ARGUMENT
        (binding-name Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-2 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para criticar un modelo de clasificador")
        (antecedent Relacion-Entre-Clasificadores-2-antec)
        (consequent Relacion-Entre-Clasificadores-2-conseq)
        (connection-symbol cambiar-a-2)
)

(make-instance Relacion-Entre-Clasificadores-3-antec of ARGUMENT
        (binding-name Violaciones-Clasificador Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-3-consec of ARGUMENT
        (binding-name Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-3 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para criticar un modelo de clasificador")
        (antecedent Relacion-Entre-Clasificadores-3-antec)
        (consequent Relacion-Entre-Clasificadores-3-conseq)
        (connection-symbol cambiar-a-3)
)

(make-instance Relacion-Entre-Clasificadores-4-antec of ARGUMENT
        (binding-name Violaciones-Clasificador Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-4-consec of ARGUMENT
        (binding-name Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Clasificadores-4 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para criticar un modelo de clasificador")
        (antecedent Relacion-Entre-Clasificadores-4-antec)
        (consequent Relacion-Entre-Clasificadores-4-conseq)
        (connection-symbol cambiar-a-4)
)

(make-instance Relacion-Entre-Patrones-antec of ARGUMENT
        (binding-name Informacion-Patrones)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Patrones-consec of ARGUMENT
        (binding-name Informacion-Patrones)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Entre-Patrones of IMPLICATION-RULETYPE
        (description "Tipo de Regla para criticar los patrones de entrada")
```

```
        (antecedent Relacion-Entre-Patrones-antec)
        (consequent Relacion-Entre-Patrones-conseq)
        (connection-symbol cambiar-a)
)

(make-instance Relacion-Ordenacion-1-antec of ARGUMENT
        (binding-name Informacion-Patrones Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-1-consec of ARGUMENT
        (binding-name Lineal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-1 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para ordenar los clasificadores válidos")
        (antecedent Relacion-Ordenacion-1-antec)
        (consequent Relacion-Ordenacion-1-conseq)
        (connection-symbol ordenar-como-1)
)

(make-instance Relacion-Ordenacion-2-antec of ARGUMENT
        (binding-name Informacion-Patrones Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-2-consec of ARGUMENT
        (binding-name Cuadratico)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-2 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para ordenar los clasificadores válidos")
        (antecedent Relacion-Ordenacion-2-antec)
        (consequent Relacion-Ordenacion-2-conseq)
        (connection-symbol ordenar-como-2)
)

(make-instance Relacion-Ordenacion-3-antec of ARGUMENT
        (binding-name Informacion-Patrones Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-3-consec of ARGUMENT
        (binding-name Red-Neuronal)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-3 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para ordenar los clasificadores válidos")
        (antecedent Relacion-Ordenacion-3-antec)
        (consequent Relacion-Ordenacion-3-conseq)
        (connection-symbol ordenar-como-3)
)

(make-instance Relacion-Ordenacion-4-antec of ARGUMENT
        (binding-name Informacion-Patrones Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-4-consec of ARGUMENT
        (binding-name Vecinos-Proximos)
        (cardinality-min 1)
        (cardinality-max 1)
        )

(make-instance Relacion-Ordenacion-4 of IMPLICATION-RULETYPE
        (description "Tipo de Regla para ordenar los clasificadores válidos")
        (antecedent Relacion-Ordenacion-4-antec)
        (consequent Relacion-Ordenacion-4-conseq)
        (connection-symbol ordenar-como-4)
)
```

LISTADO 19: Código CLIPS para el Problema del Diseño de Clasificadores

```
;;; Created with scml v1.0

(make-instance Restricciones-Diseno of DYNAMIC-ROLE (data-type Patrones-Entrada))

(make-instance Restricciones-Explicitas of DYNAMIC-ROLE (data-type Informacion-Patrones))

(make-instance Estructura-Diseno of DYNAMIC-ROLE (data-type Clasificador))

(make-instance Propuesta-Diseno-1 of STATIC-ROLE (domain-mapping Dependencia-Clasificador-Patrones-1 FROM
Propuestas-Modelos-Clasificadores-1))

(make-instance Propuesta-Diseno-2 of STATIC-ROLE (domain-mapping Dependencia-Clasificador-Patrones-2 FROM
Propuestas-Modelos-Clasificadores-2))

(make-instance Propuesta-Diseno-3 of STATIC-ROLE (domain-mapping Dependencia-Clasificador-Patrones-3 FROM
Propuestas-Modelos-Clasificadores-3))

(make-instance Propuesta-Diseno-4 of STATIC-ROLE (domain-mapping Dependencia-Clasificador-Patrones-4 FROM
Propuestas-Modelos-Clasificadores-4))

(make-instance Diseno of DYNAMIC-ROLE (data-type Clasificador))

(make-instance Prestaciones-Diseno of DYNAMIC-ROLE (data-type Prestaciones-Clasificador))

(make-instance Violaciones-Diseno of DYNAMIC-ROLE (data-type Violaciones-Clasificador))

(make-instance Valor-Verdad of DYNAMIC-ROLE (data-type TRUTH-VALUE))

(make-instance Disenos-Validos of DYNAMIC-ROLE (data-type LIST))

(make-instance Critica-Diseno-1 of STATIC-ROLE (domain-mapping Relacion-Entre-Clasificadores-1 FROM
Criticas-Modelos-Clasificadores-1))

(make-instance Critica-Diseno-2 of STATIC-ROLE (domain-mapping Relacion-Entre-Clasificadores-2 FROM
Criticas-Modelos-Clasificadores-2))

(make-instance Critica-Diseno-3 of STATIC-ROLE (domain-mapping Relacion-Entre-Clasificadores-3 FROM
Criticas-Modelos-Clasificadores-3))

(make-instance Critica-Diseno-4 of STATIC-ROLE (domain-mapping Relacion-Entre-Clasificadores-4 FROM
Criticas-Modelos-Clasificadores-4))

(make-instance Modificaciones-Diseno of DYNAMIC-ROLE (data-type Clasificador))

(make-instance Modificaciones-Restricciones of DYNAMIC-ROLE (data-type Informacion-Patrones))

(make-instance Critica-Restricciones of STATIC-ROLE (domain-mapping Relacion-Entre-Patrones FROM
Criticas-Patrones-Entrada))

(make-instance Criterio-Ordenacion-1 of STATIC-ROLE (domain-mapping Relacion-Ordenacion-1 FROM Criterios-
Ordenacion-Clasificadores-1))

(make-instance Criterio-Ordenacion-2 of STATIC-ROLE (domain-mapping Relacion-Ordenacion-2 FROM Criterios-
Ordenacion-Clasificadores-2))

(make-instance Criterio-Ordenacion-3 of STATIC-ROLE (domain-mapping Relacion-Ordenacion-3 FROM Criterios-
Ordenacion-Clasificadores-3))

(make-instance Criterio-Ordenacion-4 of STATIC-ROLE (domain-mapping Relacion-Ordenacion-4 FROM Criterios-
Ordenacion-Clasificadores-4))

(make-instance Lista-Disenos of DYNAMIC-ROLE (data-type LIST))

(make-instance Explicitar-Restricciones of INFERENCE
        (input-roles Restricciones-Diseno)
        (output-roles Restricciones-Explicitas)
        (method Explicitar-Restricciones-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
```

```
        (specification "Hace explícitas las restricciones")
        )

(make-instance Especificar-Estructura-Diseno of INFERENCE
        (input-roles Restricciones-Explicitas)
        (output-roles Estructura-Diseno)
        (static-roles Propuesta-Diseno-1 Propuesta-Diseno-2 Propuesta-Diseno-3 Propuesta-Diseno-4)
        (method Especificar-Estructura-Diseno-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "A partir de los requerimientos explícitos, especifica una estructura de Diseno")
        )

(defmodule Especificar-Estructura-Diseno-MODULE (import MAIN ?ALL) (export ?ALL))

(deftemplate Especificar-Estructura-Diseno-MODULE::consecuente-template
        (slot Tipo-Discriminante)
        (slot Estimacion-Parametros)
        (slot Numero-Entradas)
        (slot Numero-Salidas)
        (slot Priors-Clases)
        (slot Numero-Patrones-Entrenamiento)
        (slot Numero-Patrones-Validacion)
        (slot Estimacion-Error)
        (slot Error-Clasificacion)
        (slot Indice-Ordenacion)
)

; Esta función evaluará los resultados de los consecuentes y creará los hechos
; que se desprendan de estas evaluaciones.
; Para la modificación o inclusión de los operadores de las expresiones se deben modifcar
; o incluir dentro del switch.
; Las expresiones serán del tipo:
; = atributo1 4 = atributo2 1 OR = atributo1 2 = atributo2 1 = atributo3 0 ...
(deffunction Especificar-Estructura-Diseno-MODULE::funcion-evaluar ($?expresion)
        (bind $?expresion (list $?expresion))
        (bind ?hecho "(assert (consecuente-template ")
        (while (> (list-length $?expresion) 0)
                (bind ?operador (list-nth 1 $?expresion))
                (bind $?expresion (list-rest $?expresion))
                (switch ?operador
                        (case = then
                                (bind ?operando1 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?operando2 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?hecho (str-cat ?hecho "(" ?operando1 " " ?operando2 ") "))
                        )
                        (case or then
                                (bind ?hecho (str-cat ?hecho "))"))
                                (eval ?hecho)
                                (bind ?hecho "(assert(consecuente-template ")
                        )
                        (default
                        )
                )
        )
        (bind ?hecho (str-cat ?hecho "))"))
        (eval ?hecho)
)

; Activa las reglas que se podrán ejecutar.
(defrule Especificar-Estructura-Diseno-MODULE::r-1
        ?x <- (object (is-a RULE)
                (ruletype ?y&:(= 0 (str-compare ?y (nth$ 1 (send (symbol-to-instance-name (expand$
(explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::expand] get-static-roles)))))) get-domain-mapping)))))
                (knowledge-base-name ?w&:(= 0 (str-compare ?w (nth$ 3 (send (symbol-to-instance-name
(expand$ (explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::Especificar-Estructura-Diseno] get-static-
roles)))))) get-domain-mapping)))))
                (antecedent ?p)
                (consequent ?q)
                )
        =>
        (assert (se-necesita ?p ?q))
)

; Para las reglas que se cumple el antecedente se ejecutará el consecuente.
(defrule Especificar-Estructura-Diseno-MODULE::r-2
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        (resultado ?nombre-antecedente TRUE)
```

```
        =>
        (assert (ejecutar ?nombre-consecuente))
)

; Para las reglas que se activen, activamos sus antecedentes.
(defrule Especificar-Estructura-Diseno-MODULE::r-3
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?nombre-antecedente))
                 (operator ?operator)
                 (left-operand ?left-operand)
                 (right-operand ?right-operand)
              )
        =>
        (assert (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand))
)

; Para los antecedentes activados que conozcamos los valores de sus operandos.
; mandamos a calcular su resultado
(defrule Especificar-Estructura-Diseno-MODULE::r-4
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-antec ?nombre-antecedente [ $?left-value ] ?operator [ $?right-value ]))

)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores unarios
(defrule Especificar-Estructura-Diseno-MODULE::r-5-1
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ NULL ])
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores binarios
(defrule Especificar-Estructura-Diseno-MODULE::r-5-2
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ $?right-operand ])
        (test (neq (expand$ $?right-operand) NULL))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) " " (implode$
$?right-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un número.
(defrule Especificar-Estructura-Diseno-MODULE::r-6
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un valor
; de un atributo.
(defrule Especificar-Estructura-Diseno-MODULE::r-7
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        =>
        (bind ?position (eval (sub-string 3 (- (str-index "-" ?left-operand) 1) ?left-operand)))
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$ (str-cat MAIN::
(nth$ ?position (send [MAIN::expand] get-input-roles))))))get-domain-mapping))
        )
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Si el valor del operando izquierdo de un antecedente es un antecedente lo activamos.
(defrule Especificar-Estructura-Diseno-MODULE::r-8
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
```

```
                (left-operand ?left-left-operand)
                (right-operand ?left-right-operand)
             )
        =>
        (assert (necesita-antec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un número.
(defrule Especificar-Estructura-Diseno-MODULE::r-9
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un valor
; de un atributo.
(defrule Especificar-Estructura-Diseno-MODULE::r-10
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (not (eq ?right-operand NULL)))
        (test (eq "##" (sub-string 1 2 (str-cat ?right-operand))))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
(eval (sub-string 3 (- (str-index "-" ?right-operand) 1) (str-cat ?right-operand))) (send [MAIN::expand]
get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un antecedente si este es NULL.
(defrule Especificar-Estructura-Diseno-MODULE::r-26
        (necesita-antec ?nombre-antecedente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)

; Si el valor del operando derecho de un antecedente es un antecedente lo activamos.
(defrule Especificar-Estructura-Diseno-MODULE::r-11
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a ANTECEDENT)
                (name =(symbol-to-instance-name ?right-operand))
                (operator ?right-operator)
                (left-operand ?right-left-operand)
                (right-operand ?right-right-operand)
             )
        =>
        (assert (necesita-antec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)

; Para los consecuentes que ya conozcamos sus resultados generamos los hechos que se
; deduzcan a partir de estos.
(defrule Especificar-Estructura-Diseno-MODULE::r-12
        (ejecutar ?nombre-consecuente)
        (resultado ?nombre-consecuente $?lista-valor)
        =>
        (funcion-evaluar $?lista-valor)
)

; Para las reglas en las que se necesite ejecutar el consecuente se activa el consecuente.
(defrule Especificar-Estructura-Diseno-MODULE::r-13
        (ejecutar ?nombre-consecuente)
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?nombre-consecuente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
             )
        =>
        (assert (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand))
)

; Para los consecuentes activados que tengan operandos conocidos se calculará el resultado.
(defrule Especificar-Estructura-Diseno-MODULE::r-14
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
```

```
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-
operand [ $?right-value ]))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores unarios
(defrule Especificar-Estructura-Diseno-MODULE::r-15-1
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand NULL)
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores binarios
(defrule Especificar-Estructura-Diseno-MODULE::r-15-2
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        (test (not (eq (expand$ $?right-value) NULL)))
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) " " (implode$ $?right-
value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)
; Si el consecuente es una asignación se creará una expresión que la describa.
(defrule Especificar-Estructura-Diseno-MODULE::r-16
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] eq ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ = $?left-value $?right-value)))
)

; Si el consecuente es un (or) se creará una expresión que la describa.
(defrule Especificar-Estructura-Diseno-MODULE::r-17
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] or ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value or $?right-value)))
)

; Si el consecuente es un (and) se creará una expresión que la describa.
(defrule Especificar-Estructura-Diseno-MODULE::r-18
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] and ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value $?right-value)))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un número.
(defrule Especificar-Estructura-Diseno-MODULE::r-19
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (numberp ?left-operand))
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un valor
; de un atributo.
(defrule Especificar-Estructura-Diseno-MODULE::r-20
```

```
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        (test (not (eq ?operator eq)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un atributo.
(defrule Especificar-Estructura-Diseno-MODULE::r-21
        (necesita-consec ?nombre-consecuente ?left-operand eq ?right-operand)
        (test (eq @@ (sym-cat (sub-string (- (length ?left-operand) 1) (length ?left-operand) (str-cat
?left-operand)))))
        =>
        (bind ?nombre-var (sym-cat (sub-string 1 (- (length ?left-operand) 2) ?left-operand)))
        (assert (resultado ?left-operand ?nombre-var))
)

; Si el valor del operando izquierdo de un consecuente es un consecuente lo activamos.
(defrule Especificar-Estructura-Diseno-MODULE::r-22
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?left-operand))
                (operator ?left-operator)
                (left-operand ?left-left-operand)
                (right-operand ?left-right-operand)
              )
        =>
        (assert (necesita-consec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un número.
(defrule Especificar-Estructura-Diseno-MODULE::r-23
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un valor
; de un atributo.
(defrule Especificar-Estructura-Diseno-MODULE::r-24
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?right-operand)))))
        (test (not (eq ?right-operand NULL)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un consecuente si este es NULL.
(defrule Especificar-Estructura-Diseno-MODULE::r-27
        (necesita-consec ?nombre-consecuente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)
; Si el valor del operando derecho de un consecuente es un consecuente lo activamos.
(defrule Especificar-Estructura-Diseno-MODULE::r-25
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?right-operand))
                (operator ?right-operator)
                (left-operand ?right-left-operand)
                (right-operand ?right-right-operand)
              )
        =>
        (assert (necesita-consec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)
```

```
(make-instance Proponer-Diseno of INFERENCE
        (input-roles Restricciones-Diseno Estructura-Diseno)
        (output-roles Diseno)
        (method Proponer-Diseno-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Propone un Diseno")
        )

(make-instance Verificar-Diseno of INFERENCE
        (input-roles Diseno Prestaciones-Diseno)
        (output-roles Disenos-Validos Violaciones-Diseno Valor-Verdad)
        (method Verificar-Diseno-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Verifica el Diseno")
        )

(make-instance Criticar-Diseno of INFERENCE
        (input-roles Diseno Violaciones-Diseno)
        (output-roles Modificaciones-Diseno)
        (static-roles Critica-Diseno-1 Critica-Diseno-2 Critica-Diseno-3 Critica-Diseno-4)
        (method Criticar-Diseno-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Critica el Diseno proponiendo modificaciones")
        )

(defmodule Criticar-Diseno-MODULE (import MAIN ?ALL) (export ?ALL))

(deftemplate Criticar-Diseno-MODULE::consecuente-template
        (slot Tipo-Discriminante)
        (slot Estimacion-Parametros)
        (slot Numero-Entradas)
        (slot Numero-Salidas)
        (slot Priors-Clases)
        (slot Numero-Patrones-Entrenamiento)
        (slot Numero-Patrones-Validacion)
        (slot Estimacion-Error)
        (slot Error-Clasificacion)
        (slot Indice-Ordenacion)
)

; Esta función evaluará los resultados de los consecuentes y creará los hechos
; que se desprendan de estas evaluaciones.
; Para la modificación o inclusión de los operadores de las expresiones se deben modifcar
; o incluir dentro del switch.
; Las expresiones serán del tipo:
; = atributo1 4 = atributo2 1 OR = atributo1 2 = atributo2 1 = atributo3 0 ...
(deffunction Criticar-Diseno-MODULE::funcion-evaluar ($?expresion)
        (bind $?expresion (list $?expresion))
        (bind ?hecho "(assert (consecuente-template ")
        (while (> (list-length $?expresion) 0)
                (bind ?operador (list-nth 1 $?expresion))
                (bind $?expresion (list-rest $?expresion))
                (switch ?operador
                        (case = then
                                (bind ?operando1 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?operando2 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?hecho (str-cat ?hecho "(" ?operando1 " " ?operando2 ") "))
                        )
                        (case or then
                                (bind ?hecho (str-cat ?hecho "))"))
                                (eval ?hecho)
                                (bind ?hecho "(assert(consecuente-template ")
                        )
                        (default
                        )
                )
        )
        (bind ?hecho (str-cat ?hecho "))"))
        (eval ?hecho)
)

; Activa las reglas que se podrán ejecutar.
(defrule Criticar-Diseno-MODULE::r-1
        ?x <- (object (is-a RULE)
```

```
                (ruletype ?y&:(= 0 (str-compare ?y (nth$ 1 (send (symbol-to-instance-name (expand$
(explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::expand] get-static-roles)))))) get-domain-mapping)))))
                (knowledge-base-name ?w&:(= 0 (str-compare ?w (nth$ 3 (send (symbol-to-instance-name
(expand$ (explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::Criticar-Diseno] get-static-roles)))))) get-
domain-mapping)))))
                (antecedent ?p)
                (consequent ?q)
                )
        =>
        (assert (se-necesita ?p ?q))
)

; Para las reglas que se cumple el antecedente se ejecutará el consecuente.
(defrule Criticar-Diseno-MODULE::r-2
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        (resultado ?nombre-antecedente TRUE)
        =>
        (assert (ejecutar ?nombre-consecuente))
)

; Para las reglas que se activen, activamos sus antecedentes.
(defrule Criticar-Diseno-MODULE::r-3
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        ?x <- (object (is-a ANTECEDENT)
                (name =(symbol-to-instance-name ?nombre-antecedente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
              )
        =>
        (assert (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand))
)

; Para los antecedentes activados que conozcamos los valores de sus operandos.
; mandamos a calcular su resultado
(defrule Criticar-Diseno-MODULE::r-4
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-antec ?nombre-antecedente [ $?left-value ] ?operator [ $?right-value ]))

)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores unarios
(defrule Criticar-Diseno-MODULE::r-5-1
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ NULL ])
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores binarios
(defrule Criticar-Diseno-MODULE::r-5-2
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ $?right-operand ])
        (test (neq (expand$ $?right-operand) NULL))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) " " (implode$
$?right-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un número.
(defrule Criticar-Diseno-MODULE::r-6
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un valor
; de un atributo.
(defrule Criticar-Diseno-MODULE::r-7
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        =>
        (bind ?position (eval (sub-string 3 (- (str-index "-" ?left-operand) 1) ?left-operand)))
```

```
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$ (str-cat MAIN::
(nth$ ?position (send [MAIN::expand] get-input-roles))))))get-domain-mapping))
        )
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Si el valor del operando izquierdo de un antecedente es un antecedente lo activamos.
(defrule Criticar-Diseno-MODULE::r-8
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
                 (left-operand ?left-left-operand)
                 (right-operand ?left-right-operand)
              )
        =>
        (assert (necesita-antec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un número.
(defrule Criticar-Diseno-MODULE::r-9
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un valor
; de un atributo.
(defrule Criticar-Diseno-MODULE::r-10
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (not (eq ?right-operand NULL)))
        (test (eq "##" (sub-string 1 2 (str-cat ?right-operand))))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
(eval (sub-string 3 (- (str-index "-" ?right-operand) 1) (str-cat ?right-operand))) (send [MAIN::expand]
get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un antecedente si este es NULL.
(defrule Criticar-Diseno-MODULE::r-26
        (necesita-antec ?nombre-antecedente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)

; Si el valor del operando derecho de un antecedente es un antecedente lo activamos.
(defrule Criticar-Diseno-MODULE::r-11
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?right-operand))
                 (operator ?right-operator)
                 (left-operand ?right-left-operand)
                 (right-operand ?right-right-operand)
              )
        =>
        (assert (necesita-antec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)

; Para los consecuentes que ya conozcamos sus resultados generamos los hechos que se
; deduzcan a partir de estos.
(defrule Criticar-Diseno-MODULE::r-12
        (ejecutar ?nombre-consecuente)
        (resultado ?nombre-consecuente $?lista-valor)
        =>
        (funcion-evaluar $?lista-valor)
)

; Para las reglas en las que se necesite ejecutar el consecuente se activa el consecuente.
```

```
(defrule Criticar-Diseno-MODULE::r-13
        (ejecutar ?nombre-consecuente)
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?nombre-consecuente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
              )
        =>
        (assert (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand))
)

; Para los consecuentes activados que tengan operandos conocidos se calculará el resultado.
(defrule Criticar-Diseno-MODULE::r-14
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-
operand [ $?right-value ]))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores unarios
(defrule Criticar-Diseno-MODULE::r-15-1
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand NULL)
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores binarios
(defrule Criticar-Diseno-MODULE::r-15-2
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        (test (not (eq (expand$ $?right-value) NULL)))
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) " " (implode$ $?right-
value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)
; Si el consecuente es una asignación se creará una expresión que la describa.
(defrule Criticar-Diseno-MODULE::r-16
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] eq ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ = $?left-value $?right-value)))
)

; Si el consecuente es un (or) se creará una expresión que la describa.
(defrule Criticar-Diseno-MODULE::r-17
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] or ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value or $?right-value)))
)

; Si el consecuente es un (and) se creará una expresión que la describa.
(defrule Criticar-Diseno-MODULE::r-18
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] and ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
```

```
        (assert (resultado ?nombre-consecuente (create$ $?left-value $?right-value)))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un número.
(defrule Criticar-Diseno-MODULE::r-19
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (numberp ?left-operand))
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un valor
; de un atributo.
(defrule Criticar-Diseno-MODULE::r-20
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        (test (not (eq ?operator eq)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un atributo.
(defrule Criticar-Diseno-MODULE::r-21
        (necesita-consec ?nombre-consecuente ?left-operand eq ?right-operand)
        (test (eq @@ (sym-cat (sub-string (- (length ?left-operand) 1) (length ?left-operand) (str-cat
?left-operand)))))
        =>
        (bind ?nombre-var (sym-cat (sub-string 1 (- (length ?left-operand) 2) ?left-operand)))
        (assert (resultado ?left-operand ?nombre-var))
)

; Si el valor del operando izquierdo de un consecuente es un consecuente lo activamos.
(defrule Criticar-Diseno-MODULE::r-22
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a CONSEQUENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
                 (left-operand ?left-left-operand)
                 (right-operand ?left-right-operand)
               )
        =>
        (assert (necesita-consec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un número.
(defrule Criticar-Diseno-MODULE::r-23
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un valor
; de un atributo.
(defrule Criticar-Diseno-MODULE::r-24
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?right-operand)))))
        (test (not (eq ?right-operand NULL)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un consecuente si este es NULL.
(defrule Criticar-Diseno-MODULE::r-27
        (necesita-consec ?nombre-consecuente ?left-operand ?operator NULL)
        =>
```

```
        (assert (resultado NULL NULL))
)
; Si el valor del operando derecho de un consecuente es un consecuente lo activamos.
(defrule Criticar-Diseno-MODULE::r-25
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?right-operand))
                (operator ?right-operator)
                (left-operand ?right-left-operand)
                (right-operand ?right-right-operand)
              )
        =>
        (assert (necesita-consec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)
(make-instance Modificar-Diseno of INFERENCE
        (input-roles Modificaciones-Diseno)
        (output-roles Diseno)
        (method Modificar-Diseno-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Modifica el Diseno")
        )

(make-instance Criticar-Restricciones of INFERENCE
        (input-roles Restricciones-Explicitas)
        (output-roles Modificaciones-Restricciones)
        (static-roles Critica-Restricciones)
        (method Criticar-Restricciones-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Critica las restricciones proponiendo modificaciones")
        )

(defmodule Criticar-Restricciones-MODULE (import MAIN ?ALL) (export ?ALL))

(deftemplate Criticar-Restricciones-MODULE::consecuente-template
        (slot Numero-Patrones-Entrada)
        (slot Numero-Clases)
        (slot Numero-Caracteristicas)
        (slot Test1-Normalidad-Mono)
        (slot Test2-Normalidad-Mono)
        (slot Test1-Normalidad-Multi)
        (slot Distancia-Bhattacharyya)
        (slot Distancia-Bhattacharyya-1)
        (slot Distancia-Bhattacharyya-2)
        (slot Sesgo-Lineal)
        (slot Sesgo-Cuadratico)
        (slot Dimensionalidad-Intrinseca)
        (slot Reduccion-Dimensionalidad)
)

; Esta función evaluará los resultados de los consecuentes y creará los hechos
; que se desprendan de estas evaluaciones.
; Para la modificación o inclusión de los operadores de las expresiones se deben modifcar
; o incluir dentro del switch.
; Las expresiones serán del tipo:
; = atributo1 4 = atributo2 1 OR = atributo1 2 = atributo2 1 = atributo3 0 ...
(deffunction Criticar-Restricciones-MODULE::funcion-evaluar ($?expresion)
        (bind $?expresion (list $?expresion))
        (bind ?hecho "(assert (consecuente-template ")
        (while (> (list-length $?expresion) 0)
                (bind ?operador (list-nth 1 $?expresion))
                (bind $?expresion (list-rest $?expresion))
                (switch ?operador
                        (case = then
                                (bind ?operando1 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?operando2 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?hecho (str-cat ?hecho "(" ?operando1 " " ?operando2 ") "))
                        )
                        (case or then
                                (bind ?hecho (str-cat ?hecho "))"))
                                (eval ?hecho)
                                (bind ?hecho "(assert(consecuente-template ")
                        )
                        (default
                        )
```

```
                )
        )
        (bind ?hecho (str-cat ?hecho "))"))
        (eval ?hecho)
)

; Activa las reglas que se podrán ejecutar.
(defrule Criticar-Restricciones-MODULE::r-1
        ?x <- (object (is-a RULE)
                (ruletype ?y&:(= 0 (str-compare ?y (nth$ 1 (send (symbol-to-instance-name (expand$
(explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::expand] get-static-roles)))))) get-domain-mapping)))))
                (knowledge-base-name ?w&:(= 0 (str-compare ?w (nth$ 3 (send (symbol-to-instance-name
(expand$ (explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::Criticar-Restricciones] get-static-roles))))))
get-domain-mapping)))))
                (antecedent ?p)
                (consequent ?q)
                )
        =>
        (assert (se-necesita ?p ?q))
)

; Para las reglas que se cumple el antecedente se ejecutará el consecuente.
(defrule Criticar-Restricciones-MODULE::r-2
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        (resultado ?nombre-antecedente TRUE)
        =>
        (assert (ejecutar ?nombre-consecuente))
)

; Para las reglas que se activen, activamos sus antecedentes.
(defrule Criticar-Restricciones-MODULE::r-3
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        ?x <- (object (is-a ANTECEDENT)
                (name =(symbol-to-instance-name ?nombre-antecedente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
              )
        =>
        (assert (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand))
)

; Para los antecedentes activados que conozcamos los valores de sus operandos.
; mandamos a calcular su resultado
(defrule Criticar-Restricciones-MODULE::r-4
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-antec ?nombre-antecedente [ $?left-value ] ?operator [ $?right-value ]))

)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores unarios
(defrule Criticar-Restricciones-MODULE::r-5-1
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ NULL ])
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores binarios
(defrule Criticar-Restricciones-MODULE::r-5-2
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ $?right-operand ])
        (test (neq (expand$ $?right-operand) NULL))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) " " (implode$
$?right-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un número.
(defrule Criticar-Restricciones-MODULE::r-6
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
```

```
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un valor
; de un atributo.
(defrule Criticar-Restricciones-MODULE::r-7
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        =>
        (bind ?position (eval (sub-string 3 (- (str-index "-" ?left-operand) 1) ?left-operand)))
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$ (str-cat MAIN::
(nth$ ?position (send [MAIN::expand] get-input-roles))))))get-domain-mapping))
        )
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Si el valor del operando izquierdo de un antecedente es un antecedente lo activamos.
(defrule Criticar-Restricciones-MODULE::r-8
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
                 (left-operand ?left-left-operand)
                 (right-operand ?left-right-operand)
               )
        =>
        (assert (necesita-antec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un número.
(defrule Criticar-Restricciones-MODULE::r-9
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un valor
; de un atributo.
(defrule Criticar-Restricciones-MODULE::r-10
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (not (eq ?right-operand NULL)))
        (test (eq "##" (sub-string 1 2 (str-cat ?right-operand))))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
(eval (sub-string 3 (- (str-index "-" ?right-operand) 1) (str-cat ?right-operand))) (send [MAIN::expand]
get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un antecedente si este es NULL.
(defrule Criticar-Restricciones-MODULE::r-26
        (necesita-antec ?nombre-antecedente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)

; Si el valor del operando derecho de un antecedente es un antecedente lo activamos.
(defrule Criticar-Restricciones-MODULE::r-11
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?right-operand))
                 (operator ?right-operator)
                 (left-operand ?right-left-operand)
                 (right-operand ?right-right-operand)
               )
        =>
        (assert (necesita-antec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)

; Para los consecuentes que ya conozcamos sus resultados generamos los hechos que se
```

```
; deduzcan a partir de estos.
(defrule Criticar-Restricciones-MODULE::r-12
        (ejecutar ?nombre-consecuente)
        (resultado ?nombre-consecuente $?lista-valor)
        =>
        (funcion-evaluar $?lista-valor)
)

; Para las reglas en las que se necesite ejecutar el consecuente se activa el consecuente.
(defrule Criticar-Restricciones-MODULE::r-13
        (ejecutar ?nombre-consecuente)
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?nombre-consecuente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
              )
        =>
        (assert (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand))
)

; Para los consecuentes activados que tengan operandos conocidos se calculará el resultado.
(defrule Criticar-Restricciones-MODULE::r-14
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-
operand [ $?right-value ]))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores unarios
(defrule Criticar-Restricciones-MODULE::r-15-1
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand NULL)
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores binarios
(defrule Criticar-Restricciones-MODULE::r-15-2
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        (test (not (eq (expand$ $?right-value) NULL)))
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) " " (implode$ $?right-
value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)
; Si el consecuente es una asignación se creará una expresión que la describa.
(defrule Criticar-Restricciones-MODULE::r-16
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] eq ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ = $?left-value $?right-value)))
)

; Si el consecuente es un (or) se creará una expresión que la describa.
(defrule Criticar-Restricciones-MODULE::r-17
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] or ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value or $?right-value)))
```

```
)

; Si el consecuente es un (and) se creará una expresión que la describa.
(defrule Criticar-Restricciones-MODULE::r-18
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] and ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value $?right-value)))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un número.
(defrule Criticar-Restricciones-MODULE::r-19
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (numberp ?left-operand))
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un valor
; de un atributo.
(defrule Criticar-Restricciones-MODULE::r-20
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        (test (not (eq ?operator eq)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un atributo.
(defrule Criticar-Restricciones-MODULE::r-21
        (necesita-consec ?nombre-consecuente ?left-operand eq ?right-operand)
        (test (eq @@ (sym-cat (sub-string (- (length ?left-operand) 1) (length ?left-operand) (str-cat
?left-operand)))))
        =>
        (bind ?nombre-var (sym-cat (sub-string 1 (- (length ?left-operand) 2) ?left-operand)))
        (assert (resultado ?left-operand ?nombre-var))
)

; Si el valor del operando izquierdo de un consecuente es un consecuente lo activamos.
(defrule Criticar-Restricciones-MODULE::r-22
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?left-operand))
                (operator ?left-operator)
                (left-operand ?left-left-operand)
                (right-operand ?left-right-operand)
              )
        =>
        (assert (necesita-consec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un número.
(defrule Criticar-Restricciones-MODULE::r-23
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un valor
; de un atributo.
(defrule Criticar-Restricciones-MODULE::r-24
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?right-operand)))))
        (test (not (eq ?right-operand NULL)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
```

```
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un consecuente si este es NULL.
(defrule Criticar-Restricciones-MODULE::r-27
        (necesita-consec ?nombre-consecuente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)
; Si el valor del operando derecho de un consecuente es un consecuente lo activamos.
(defrule Criticar-Restricciones-MODULE::r-25
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?right-operand))
                (operator ?right-operator)
                (left-operand ?right-left-operand)
                (right-operand ?right-right-operand)
              )
        =>
        (assert (necesita-consec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)
(make-instance Modificar-Restricciones of INFERENCE
        (input-roles Restricciones-Diseno Modificaciones-Restricciones)
        (output-roles Restricciones-Diseno)
        (method Modificar-Restricciones-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Modifica las restricciones")
        )

(make-instance Ordenar-Disenos of INFERENCE
        (input-roles Restricciones-Explicitas Disenos-Validos)
        (output-roles Lista-Disenos)
        (static-roles Criterio-Ordenacion-1 Criterio-Ordenacion-2 Criterio-Ordenacion-3 Criterio-
Ordenacion-4)
        (method Ordenar-Disenos-method)
        (has-solution-method generic-has-solution)
        (new-solution-method generic-new-solution)
        (specification "Critica las restricciones proponiendo modificaciones")
        )

(defmodule Ordenar-Disenos-MODULE (import MAIN ?ALL) (export ?ALL))

(deftemplate Ordenar-Disenos-MODULE::consecuente-template
        (slot Tipo-Discriminante)
        (slot Estimacion-Parametros)
        (slot Numero-Entradas)
        (slot Numero-Salidas)
        (slot Priors-Clases)
        (slot Numero-Patrones-Entrenamiento)
        (slot Numero-Patrones-Validacion)
        (slot Estimacion-Error)
        (slot Error-Clasificacion)
        (slot Indice-Ordenacion)
)

; Esta función evaluará los resultados de los consecuentes y creará los hechos
; que se desprendan de estas evaluaciones.
; Para la modificación o inclusión de los operadores de las expresiones se deben modifcar
; o incluir dentro del switch.
; Las expresiones serán del tipo:
; = atributo1 4 = atributo2 1 OR = atributo1 2 = atributo2 1 = atributo3 0 ...
(deffunction Ordenar-Disenos-MODULE::funcion-evaluar ($?expresion)
        (bind $?expresion (list $?expresion))
        (bind ?hecho "(assert (consecuente-template ")
        (while (> (list-length $?expresion) 0)
                (bind ?operador (list-nth 1 $?expresion))
                (bind $?expresion (list-rest $?expresion))
                (switch ?operador
                        (case = then
                                (bind ?operando1 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?operando2 (implode$ (create$ (list-nth 1 $?expresion))))
                                (bind $?expresion (list-rest $?expresion))
                                (bind ?hecho (str-cat ?hecho "(" ?operando1 " " ?operando2 ") "))
```

```
                        )
                        (case or then
                                (bind ?hecho (str-cat ?hecho "))"))
                                (eval ?hecho)
                                (bind ?hecho "(assert(consecuente-template ")
                        )
                        (default
                        )
                )
        )
        (bind ?hecho (str-cat ?hecho "))"))
        (eval ?hecho)
)

; Activa las reglas que se podrán ejecutar.
(defrule Ordenar-Disenos-MODULE::r-1
        ?x <- (object (is-a RULE)
                (ruletype ?y&:(= 0 (str-compare ?y (nth$ 1 (send (symbol-to-instance-name (expand$
(explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::expand] get-static-roles)))))) get-domain-mapping)))))
                (knowledge-base-name ?w&:(= 0 (str-compare ?w (nth$ 3 (send (symbol-to-instance-name
(expand$ (explode$(str-cat MAIN:: (nth$ 1 (send [MAIN::Ordenar-Disenos] get-static-roles)))))) get-
domain-mapping)))))
                (antecedent ?p)
                (consequent ?q)
                )
        =>
        (assert (se-necesita ?p ?q))
)

; Para las reglas que se cumple el antecedente se ejecutará el consecuente.
(defrule Ordenar-Disenos-MODULE::r-2
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        (resultado ?nombre-antecedente TRUE)
        =>
        (assert (ejecutar ?nombre-consecuente))
)

; Para las reglas que se activen, activamos sus antecedentes.
(defrule Ordenar-Disenos-MODULE::r-3
        (se-necesita ?nombre-antecedente ?nombre-consecuente)
        ?x <- (object (is-a ANTECEDENT)
                (name =(symbol-to-instance-name ?nombre-antecedente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
              )
        =>
        (assert (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand))
)

; Para los antecedentes activados que conozcamos los valores de sus operandos.
; mandamos a calcular su resultado
(defrule Ordenar-Disenos-MODULE::r-4
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-antec ?nombre-antecedente [ $?left-value ] ?operator [ $?right-value ]))

)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores unarios
(defrule Ordenar-Disenos-MODULE::r-5-1
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ NULL ])
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)

; Calculamos el resultado de los antecedentes que se necesitan.
; Operadores binarios
(defrule Ordenar-Disenos-MODULE::r-5-2
        (calcular-antec ?nombre-antecedente [ $?left-operand ] ?operator [ $?right-operand ])
        (test (neq (expand$ $?right-operand) NULL))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-operand) " " (implode$
$?right-operand) ")")))
        (assert (resultado ?nombre-antecedente ?resultado))
)
```

```
; Calculamos el valor del operando izquierdo de un antecedente si este es un número.
(defrule Ordenar-Disenos-MODULE::r-6
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un antecedente si este es un valor
; de un atributo.
(defrule Ordenar-Disenos-MODULE::r-7
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        =>
        (bind ?position (eval (sub-string 3 (- (str-index "-" ?left-operand) 1) ?left-operand)))
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$ (str-cat MAIN::
(nth$ ?position (send [MAIN::expand] get-input-roles))))))get-domain-mapping))
        )
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Si el valor del operando izquierdo de un antecedente es un antecedente lo activamos.
(defrule Ordenar-Disenos-MODULE::r-8
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
                 (left-operand ?left-left-operand)
                 (right-operand ?left-right-operand)
               )
        =>
        (assert (necesita-antec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un número.
(defrule Ordenar-Disenos-MODULE::r-9
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un antecedente si este es un valor
; de un atributo.
(defrule Ordenar-Disenos-MODULE::r-10
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (not (eq ?right-operand NULL)))
        (test (eq "##" (sub-string 1 2 (str-cat ?right-operand))))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
(eval (sub-string 3 (- (str-index "-" ?right-operand) 1) (str-cat ?right-operand))) (send [MAIN::expand]
get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un antecedente si este es NULL.
(defrule Ordenar-Disenos-MODULE::r-26
        (necesita-antec ?nombre-antecedente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)

; Si el valor del operando derecho de un antecedente es un antecedente lo activamos.
(defrule Ordenar-Disenos-MODULE::r-11
        (necesita-antec ?nombre-antecedente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a ANTECEDENT)
                 (name =(symbol-to-instance-name ?right-operand))
                 (operator ?right-operator)
                 (left-operand ?right-left-operand)
```

```
                (right-operand ?right-right-operand)
             )
        =>
        (assert (necesita-antec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)

; Para los consecuentes que ya conozcamos sus resultados generamos los hechos que se
; deduzcan a partir de estos.
(defrule Ordenar-Disenos-MODULE::r-12
        (ejecutar ?nombre-consecuente)
        (resultado ?nombre-consecuente $?lista-valor)
        =>
        (funcion-evaluar $?lista-valor)
)

; Para las reglas en las que se necesite ejecutar el consecuente se activa el consecuente.
(defrule Ordenar-Disenos-MODULE::r-13
        (ejecutar ?nombre-consecuente)
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?nombre-consecuente))
                (operator ?operator)
                (left-operand ?left-operand)
                (right-operand ?right-operand)
             )
        =>
        (assert (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand))
)

; Para los consecuentes activados que tengan operandos conocidos se calculará el resultado.
(defrule Ordenar-Disenos-MODULE::r-14
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-
operand [ $?right-value ]))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores unarios
(defrule Ordenar-Disenos-MODULE::r-15-1
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand NULL)
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)

; Si el consecuente es una expresión aritmética se calcula su resultado.
; Operadores binarios
(defrule Ordenar-Disenos-MODULE::r-15-2
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] ?operator ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        (test (not (eq (expand$ $?right-value) NULL)))
        (test (not (eq ?operator eq)))
        (test (not (eq ?operator and)))
        (test (not (eq ?operator or)))
        =>
        (bind ?resultado (eval (str-cat "(" ?operator " " (implode$ $?left-value) " " (implode$ $?right-
value) ")")))
        (assert (resultado ?nombre-consecuente ?resultado))
)
; Si el consecuente es una asignación se creará una expresión que la describa.
(defrule Ordenar-Disenos-MODULE::r-16
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] eq ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ = $?left-value $?right-value)))
)
```

```
; Si el consecuente es un (or) se creará una expresión que la describa.
(defrule Ordenar-Disenos-MODULE::r-17
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] or ?right-operand [ $?right-
value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value or $?right-value)))
)

; Si el consecuente es un (and) se creará una expresión que la describa.
(defrule Ordenar-Disenos-MODULE::r-18
        (calcular-consec ?nombre-consecuente ?left-operand [ $?left-value ] and ?right-operand [
$?right-value ])
        (resultado ?left-operand $?left-value)
        (resultado ?right-operand $?right-value)
        =>
        (assert (resultado ?nombre-consecuente (create$ $?left-value $?right-value)))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un número.
(defrule Ordenar-Disenos-MODULE::r-19
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (numberp ?left-operand))
        (test (and (not (instance-existp (sym-cat MAIN:: ?left-operand))) (neq ## (sym-cat (sub-string 1
2 ?left-operand)))))
        =>
        (assert (resultado ?left-operand ?left-operand))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un valor
; de un atributo.
(defrule Ordenar-Disenos-MODULE::r-20
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?left-operand)))))
        (test (not (eq ?operator eq)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?left-operand)) (str-length ?left-operand) ?left-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?left-operand ?resultado))
)

; Calculamos el valor del operando izquierdo de un consecuente si este es un atributo.
(defrule Ordenar-Disenos-MODULE::r-21
        (necesita-consec ?nombre-consecuente ?left-operand eq ?right-operand)
        (test (eq @@ (sym-cat (sub-string (- (length ?left-operand) 1) (length ?left-operand) (str-cat
?left-operand)))))
        =>
        (bind ?nombre-var (sym-cat (sub-string 1 (- (length ?left-operand) 2) ?left-operand)))
        (assert (resultado ?left-operand ?nombre-var))
)

; Si el valor del operando izquierdo de un consecuente es un consecuente lo activamos.
(defrule Ordenar-Disenos-MODULE::r-22
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?left-operand)))
        ?x <- (object (is-a CONSEQUENT)
                 (name =(symbol-to-instance-name ?left-operand))
                 (operator ?left-operator)
                 (left-operand ?left-left-operand)
                 (right-operand ?left-right-operand)
              )
        =>
        (assert (necesita-consec ?left-operand ?left-left-operand ?left-operator ?left-right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un número.
(defrule Ordenar-Disenos-MODULE::r-23
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (and (not (instance-existp (sym-cat MAIN:: ?right-operand))) (neq ## (sym-cat (sub-string
1 2 (str-cat ?right-operand))))))
        =>
        (assert (resultado ?right-operand ?right-operand))
)

; Calculamos el valor del operando derecho de un consecuente si este es un valor
; de un atributo.
```

```
(defrule Ordenar-Disenos-MODULE::r-24
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (eq ## (sym-cat (sub-string 1 2 (str-cat ?right-operand)))))
        (test (not (eq ?right-operand NULL)))
        =>
        (bind ?input-role (nth$ 1 (send (symbol-to-instance-name (expand$ (explode$(str-cat MAIN:: (nth$
1 (send [MAIN::expand] get-input-roles)))))) get-domain-mapping)))
        (bind ?comando (str-cat "( send [MAIN::" ?input-role "] get-" (sub-string (+ 1 (str-index "-"
?right-operand)) (str-length ?right-operand) ?right-operand) ")" ))
        (bind ?resultado (eval ?comando))
        (assert (resultado ?right-operand ?resultado))
)

; Calculamos el valor del operando derecho de un consecuente si este es NULL.
(defrule Ordenar-Disenos-MODULE::r-27
        (necesita-consec ?nombre-consecuente ?left-operand ?operator NULL)
        =>
        (assert (resultado NULL NULL))
)
; Si el valor del operando derecho de un consecuente es un consecuente lo activamos.
(defrule Ordenar-Disenos-MODULE::r-25
        (necesita-consec ?nombre-consecuente ?left-operand ?operator ?right-operand)
        (test (instance-existp (sym-cat MAIN:: ?right-operand)))
        ?x <- (object (is-a CONSEQUENT)
                (name =(symbol-to-instance-name ?right-operand))
                (operator ?right-operator)
                (left-operand ?right-left-operand)
                (right-operand ?right-right-operand)
              )
        =>
        (assert (necesita-consec ?right-operand ?right-left-operand ?right-operator ?right-right-
operand))
)
```

LISTADO 20: Código CLIPS para el Problema del Diseño de Clasificadores

```
;;; Created with scml v1.0

(make-instance Diseno-Clasificador of TASK
        (goal "Diseñar un clasificador")
        (I-roles Restricciones-Diseno Prestaciones-Diseno)
        (O-roles Lista-Disenos)
        (method Diseno-Usando-Proponer-Criticar-Modificar)
)
```

LISTADO 21: Código R para el Problema del Diseño de Clasificadores

```
library(MASS)
library(mva)
library(nnet)
library(class)

#IRIS
data(iris)
ir <- as.matrix(iris[, -5])
#ir <- scale(ir)
tp <- factor(iris$Species)
error.ir.lda.total <- 0
error.ir.lda.t.total <- 0
error.ir.lda.deb.total <- 0
error.ir.lda.pred.total <- 0
```

```
error.ir.qda.total <- 0
error.ir.qda.t.total <- 0
error.ir.qda.deb.total <- 0
error.ir.qda.pred.total <- 0
error.ir.nnet.total.p <- 0
error.ir.nnet.total.2p <- 0
error.ir.nnet.total.3p <- 0
error.ir.nnet.001.total.p <- 0
error.ir.nnet.001.total.2p <- 0
error.ir.nnet.001.total.3p <- 0
error.ir.nnet.01.total.p <- 0
error.ir.nnet.01.total.2p <- 0
error.ir.nnet.01.total.3p <- 0
error.ir.nnet.1.total.p <- 0
error.ir.nnet.1.total.2p <- 0
error.ir.nnet.1.total.3p <- 0
error.ir.knn.1.total <- 0
error.ir.knn.2.total <- 0
error.ir.knn.3.total <- 0

for (i in 1:nrow(ir))
{
#train.clase1 <- sample(1:50, (2/3)*50)
#train.clase2 <- sample(51:100, (2/3)*50)
#train.clase3 <- sample(101:150, (2/3)*50)
#train <- c(train.clase1,train.clase2,train.clase3)
#train <- sample(1:nrow(ir), (2/3)*nrow(ir))
train <- 1:nrow(ir)
train <- train[-i]

ir.lda  <- lda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.lda, ir[-train, ])
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.lda <- error1+error2+error3
error.ir.lda.total <- error.ir.lda.total + error.ir.lda

ir.lda  <- lda(ir[train, ], tp[train], method='t')
prediccion.ir <- predict(ir.lda, ir[-train, ])
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.lda.t <- error1+error2+error3
error.ir.lda.t.total <- error.ir.lda.t.total + error.ir.lda.t

#ir.lda  <- lda(ir[train, ], tp[train], method='mve')
#prediccion.ir <- predict(ir.lda, ir[-train, ])
#indice1 <- iris[-train, ]$Species == 'setosa'
#indice2 <- iris[-train, ]$Species == 'versicolor'
#indice3 <- iris[-train, ]$Species == 'virginica'
#error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
#error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
#error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
#error.ir.lda.mve <- error1+error2+error3

ir.lda  <- lda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.lda, ir[-train, ], method='debiased')
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
```

```
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.lda.deb <- error1+error2+error3
error.ir.lda.deb.total <- error.ir.lda.deb.total +error.ir.lda.deb

ir.lda  <- lda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.lda, ir[-train, ], method='predictive')
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.lda.pred <- error1+error2+error3
error.ir.lda.pred.total <- error.ir.lda.pred.total +error.ir.lda.pred

ir.qda  <- qda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.qda, ir[-train, ])
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.qda <- error1+error2+error3
error.ir.qda.total <- error.ir.qda.total +error.ir.qda

ir.qda  <- qda(ir[train, ], tp[train], method='t')
prediccion.ir <- predict(ir.qda, ir[-train, ])
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.qda.t <- error1+error2+error3
error.ir.qda.t.total <- error.ir.qda.t.total +error.ir.qda.t

#ir.qda  <- qda(ir[train, ], tp[train], method='mve')
#prediccion.ir <- predict(ir.qda, ir[-train, ])
#indice1 <- iris[-train, ]$Species == 'setosa'
#indice2 <- iris[-train, ]$Species == 'versicolor'
#indice3 <- iris[-train, ]$Species == 'virginica'
#error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
#error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
#error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
#error.ir.qda.mve <- error1+error2+error3

ir.qda  <- qda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.qda, ir[-train, ], method='debiased')
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.qda.deb <- error1+error2+error3
error.ir.qda.deb.total <- error.ir.qda.deb.total +error.ir.qda.deb

ir.qda  <- qda(ir[train, ], tp[train])
prediccion.ir <- predict(ir.qda, ir[-train, ], method='predictive')
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
```

```
error1 <- sum(prediccion.ir$class[indice1] == 'versicolor' | prediccion.ir$class[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir$class[indice2] == 'setosa' | prediccion.ir$class[indice2] ==
'virginica')/length(indice2)
error3 <- sum(prediccion.ir$class[indice3] == 'setosa' | prediccion.ir$class[indice3] ==
'versicolor')/length(indice3)
error.ir.qda.pred <- error1+error2+error3
error.ir.qda.pred.total <- error.ir.qda.pred.total +error.ir.qda.pred

p <- 4
tpi <- class.ind(tp)
ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = p, decay = 0, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.p <- error1+error2+error3
error.ir.nnet.total.p <- error.ir.nnet.total.p +error.ir.nnet.p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 2*p, decay = 0, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.2p <- error1+error2+error3
error.ir.nnet.total.2p <- error.ir.nnet.total.2p +error.ir.nnet.2p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 3*p, decay = 0, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.3p <- error1+error2+error3
error.ir.nnet.total.3p <- error.ir.nnet.total.3p +error.ir.nnet.3p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = p, decay = 0.001, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.001.p <- error1+error2+error3
error.ir.nnet.001.total.p <- error.ir.nnet.001.total.p +error.ir.nnet.001.p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = p, decay = 0.01, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.01.p <- error1+error2+error3
error.ir.nnet.01.total.p <- error.ir.nnet.01.total.p +error.ir.nnet.01.p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = p, decay = 0.1, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
```

```
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.1.p <- error1+error2+error3
error.ir.nnet.1.total.p <- error.ir.nnet.1.total.p +error.ir.nnet.1.p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 2*p, decay = 0.001, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.001.2p <- error1+error2+error3
error.ir.nnet.001.total.2p <- error.ir.nnet.001.total.2p +error.ir.nnet.001.2p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 2*p, decay = 0.01, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.01.2p <- error1+error2+error3
error.ir.nnet.01.total.2p <- error.ir.nnet.01.total.2p +error.ir.nnet.01.2p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 2*p, decay = 0.1, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.1.2p <- error1+error2+error3
error.ir.nnet.1.total.2p <- error.ir.nnet.1.total.2p +error.ir.nnet.1.2p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 3*p, decay = 0.001, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.001.3p <- error1+error2+error3
error.ir.nnet.001.total.3p <- error.ir.nnet.001.total.3p +error.ir.nnet.001.3p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 3*p, decay = 0.01, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.01.3p <- error1+error2+error3
error.ir.nnet.01.total.3p <- error.ir.nnet.01.total.3p +error.ir.nnet.01.3p

ir.nnet <- nnet(ir[train, ], tpi[train, ], softmax = T, size = 3*p, decay = 0.1, maxit = 1000)
prediccion.ir <- predict(ir.nnet, ir[-train, ], type="class")
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
```

```
error1 <- sum(prediccion.ir[indice1] == 'versicolor' | prediccion.ir[indice1] ==
'virginica')/length(indice1)
error2 <- sum(prediccion.ir[indice2] == 'setosa' | prediccion.ir[indice2] == 'virginica')/length(indice2)
error3 <- sum(prediccion.ir[indice3] == 'setosa' | prediccion.ir[indice3] ==
'versicolor')/length(indice3)
error.ir.nnet.1.3p <- error1+error2+error3
error.ir.nnet.1.total.3p <- error.ir.nnet.1.total.3p +error.ir.nnet.1.3p

ir.knn <- knn(ir[train, ], ir[-train, ], tp[train], k = 1, prob = T)
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(ir.knn[indice1] == 'versicolor' | ir.knn[indice1] == 'virginica')/length(indice1)
error2 <- sum(ir.knn[indice2] == 'setosa' | ir.knn[indice2] == 'virginica')/length(indice2)
error3 <- sum(ir.knn[indice3] == 'setosa' | ir.knn[indice3] == 'versicolor')/length(indice3)
error.ir.knn.1 <- error1+error2+error3
error.ir.knn.1.total <- error.ir.knn.1.total +error.ir.knn.1

ir.knn <- knn(ir[train, ], ir[-train, ], tp[train], k = 2, prob = T)
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(ir.knn[indice1] == 'versicolor' | ir.knn[indice1] == 'virginica')/length(indice1)
error2 <- sum(ir.knn[indice2] == 'setosa' | ir.knn[indice2] == 'virginica')/length(indice2)
error3 <- sum(ir.knn[indice3] == 'setosa' | ir.knn[indice3] == 'versicolor')/length(indice3)
error.ir.knn.2 <- error1+error2+error3
error.ir.knn.2.total <- error.ir.knn.2.total +error.ir.knn.2

ir.knn <- knn(ir[train, ], ir[-train, ], tp[train], k = 3, prob = T)
indice1 <- iris[-train, ]$Species == 'setosa'
indice2 <- iris[-train, ]$Species == 'versicolor'
indice3 <- iris[-train, ]$Species == 'virginica'
error1 <- sum(ir.knn[indice1] == 'versicolor' | ir.knn[indice1] == 'virginica')/length(indice1)
error2 <- sum(ir.knn[indice2] == 'setosa' | ir.knn[indice2] == 'virginica')/length(indice2)
error3 <- sum(ir.knn[indice3] == 'setosa' | ir.knn[indice3] == 'versicolor')/length(indice3)
error.ir.knn.3 <- error1+error2+error3
error.ir.knn.3.total <- error.ir.knn.3.total +error.ir.knn.3
}
error.ir.lda.total <- error.ir.lda.total/nrow(ir)
error.ir.lda.t.total <- error.ir.lda.t.total/nrow(ir)
error.ir.lda.deb.total <- error.ir.lda.deb.total/nrow(ir)
error.ir.lda.pred.total <- error.ir.lda.pred.total/nrow(ir)
error.ir.qda.total <- error.ir.qda.total/nrow(ir)
error.ir.qda.t.total <- error.ir.qda.t.total/nrow(ir)
error.ir.qda.deb.total <- error.ir.qda.deb.total/nrow(ir)
error.ir.qda.pred.total <- error.ir.qda.pred.total/nrow(ir)
error.ir.nnet.total.p <- error.ir.nnet.total.p/nrow(ir)
error.ir.nnet.total.2p <- error.ir.nnet.total.2p/nrow(ir)
error.ir.nnet.total.3p <- error.ir.nnet.total.3p/nrow(ir)
error.ir.nnet.001.total.p <- error.ir.nnet.001.total.p/nrow(ir)
error.ir.nnet.001.total.2p <- error.ir.nnet.001.total.2p/nrow(ir)
error.ir.nnet.001.total.3p <- error.ir.nnet.001.total.3p/nrow(ir)
error.ir.nnet.01.total.p <- error.ir.nnet.01.total.p/nrow(ir)
error.ir.nnet.01.total.2p <- error.ir.nnet.01.total.2p/nrow(ir)
error.ir.nnet.01.total.3p <- error.ir.nnet.01.total.3p/nrow(ir)
error.ir.nnet.1.total.p <- error.ir.nnet.1.total.p/nrow(ir)
error.ir.nnet.1.total.2p <- error.ir.nnet.1.total.2p/nrow(ir)
error.ir.nnet.1.total.3p <- error.ir.nnet.1.total.3p/nrow(ir)
error.ir.knn.1.total <- error.ir.knn.1.total/nrow(ir)
error.ir.knn.2.total <- error.ir.knn.2.total/nrow(ir)
error.ir.knn.3.total <- error.ir.knn.3.total/nrow(ir)
```

LISTADO 22: Sintaxis BNF de la Versión Reducida de CML

```
program                                 ::=     KNOWLEDGE-MODEL ID  ';'
                                                [includePart]
                                                domainKnowledgeDefinitions
                                                inferenceKnowledgeDefinitions
                                                taskKnowledgeDefinitions
                                                END KNOWLEDGE-MODEL ID ';'
```

```
                                        |   [includePart]
                                            domainKnowledgeDefinitions
                                        |   [includePart]
                                            domainSchemaDefinitions
                                        |   [includePart]
                                            knowledgeBaseDefinitions
                                        |   [includePart]
                                            inferenceKnowledgeDefinitions
                                        |   [includePart]
                                            taskKnowledgeDefinitions

includePart                         ::=     includeDefinition +

includeDefinition                   ::=     INCLUDE includeList ';'

includeList                         ::=     includeItem +

includeItem                         ::=     CADENA [WITH-CODE CADENA]
                                        |   COMPILED CADENA

domainKnowledgeDefinitions          ::=     domainKnowledgeDefinition +

domainKnowledgeDefinition ::=           DOMAIN-KNOWLEDGE ID ';'
                                            [includePart]
                                            domainSchemaDefinitions
                                            knowledgeBaseDefinitions
                                            END DOMAIN-KNOWLEDGE ID ';'

DomainSchemaDefinitions             ::=     domainSchemaDefinition +

KnowledgeBaseDefinitions            ::=     knowledgeBaseDefinition +

DomainSchemaDefinition              ::=     DOMAIN-SCHEMA ID ';'
                                            [includePart]
                                            conceptDefinitions
                                            [ruletypeDefinitions]
                                            END DOMAIN-SCHEMA ID ';'

ConceptDefinitions                  ::=     conceptDefinition +

ConceptDefinition                   ::=     CONCEPT ID ';'
                                            DESCRIPTION ':' CADENA ';'
                                    [SUBTYPE-OF ':' ID ';']
                                            ATTRIBUTES ':' attributeDefinitions
                                            END CONCEPT ID ';'

AttributeDefinitions                ::=     attributeDefinition +

AttributeDefinition                 ::=     ID ':' attributeType ';'
                                            [cardinalityDefinition]

cardinalityDefinition               ::=     CARDINALITY ':' cardinalitySpec ';'
attributeType                       ::=     STRING
                                        |   NATURAL
                                        |   COLLECTION

cardinalitySpec                     ::=     CONST ['+']
                                        |   CONST '-' CONST
                                        |   ANY

ruletypeDefinitions                 ::=     ruletypeDefinition +

ruletypeDefinition                  ::=     RULE-TYPE ID ';'
                                            DESCRIPTION ':' CADENA ';'
                                    ruletypeBody
                                            END RULE-TYPE ID ';'

ruletypeBody                        ::=     implicationRuletype
                                        |   constraintRuletype

implicationRuletype                 ::=     ANTECEDENT ':' IDList ';'
                                            [cardinalityDefinition]
                                    CONSEQUENT ':' ID ';'
                                            [cardinalityDefinition]
                                    CONNECTION-SYMBOL ':' ID ';'

constraintRuletype                  ::=     CONSTRAINT        ':' IDList ';'
                                            [cardinalityDefinition]

IDList                              ::=     ID [, …]
```

```
knowledgeBaseDefinition                    ::=          KNOWLEDGE-BASE ID ';'
                                                        [includePart]
                                                        USES ':' usesDefinition ';'
                                           knowledgeComponents
                                                        END KNOWLEDGE-BASE ID ';'

usesDefinition                             ::=          ID [FROM ID]

knowledgeComponents                        ::=          [instancesComponents]
                                                        [expressionsComponents]

expressionsComponents                      ::=          EXPRESSIONS ':' ruleDefinitions

ruleDefinitions                            ::=          ruleDefinition *

ruleDefinition                             ::=          ruleExpression '==' ID
                                                        '=>' ruleExpression ';'

ruleExpression                             ::=          binaryRuleExpression
                                           |        ID '(' ruleExpression
                                                        [, ruleExpression] ')'
                                                    |   '(' ruleExpression ')'
                                                    |   symbolValue
                                                    |   symbolAttribute

binaryRuleExpression                       ::=          ruleExpression OR ruleExpression
                                                    |   ruleExpression AND ruleExpression
                                                    |   ruleExpression '=' ruleExpression
                                                    |   ruleExpression '<>' ruleExpression
                                                    |   ruleExpression '<' ruleExpression
                                                    |   ruleExpression '>' ruleExpression
                                                    |   ruleExpression '>=' ruleExpression
                                                    |   ruleExpression '<=' ruleExpression
                                                    |   ruleExpression '+' ruleExpression
                                                    |   ruleExpression '-' ruleExpression
                                                    |   ruleExpression '*' ruleExpression
                                                    |   ruleExpression '/' ruleExpression
                  |       ruleExpression '%' ruleExpression

symbolValue                                ::=          ID
                                                    |   LISTVALUE
                                                    | number
                                                    | CADENA

symbolAttribute                            ::=          ID [indice] '.' ID

indice                                     ::=          '[' number ']'

number                                     ::=          ['+' | '-'] CONST

instancesComponents                        ::=          [INSTANCES ':'] instanceDefinitions

instanceDefinitions                        ::=          instanceDefinition +

instanceDefinition                         ::=          INSTANCE ID ';'
                                                        INSTANCE-OF ':' ID ';'
                                                        ATTRIBUTES ':' attributesInstances
                                                        END INSTANCE ID ';'

attributesInstances                        ::=          attributesInstance +

attributesInstance                         ::=          ID ':' symbolValue ';'

inferenceKnowledgeDefinitions              ::= inferenceKnowledgeDefinition +

inferenceKnowledgeDefinition ::=           INFERENCE-KNOWLEDGE ID ';'
                                                        [includePart]
                                                        [roleDefinitions]
                                                        [inferenceDefinitions]
                                                        END INFERENCE-KNOWLEDGE ID ';'

roleDefinitions                            ::=          roleDefinition +

roleDefinition                             ::=          KNOWLEDGE-ROLE ID ';'
                                                        TYPE ':' typeRole ';'
                                                        DOMAIN-MAPPING ':'
                                                        mappingDefinition ';'
                                           END KNOWLEDGE-ROLE ID ';'
```

```
typeRole                                ::=          STATIC
                                           |         DYNAMIC

mappingDefinition                       ::=          ID [FROM ID]

inferenceDefinitions                    ::=          inferenceDefinition +

inferenceDefinition                     ::=          INFERENCE ID ';'
                                                     ROLES ':'
                        INPUT ':' IDList ';'
                                                      OUTPUT ':' IDList ';'
                                      [STATIC ':' IDList ';']
                                                     [METHOD ':' ID ';']
                                                     [HAS-SOLUTION ':' ID ';']
                                                     [NEW-SOLUTION ':' ID ';']
                                                     [SPECIFICATION ':' CADENA ';']
                                                     [codePart]
                                                     END INFERENCE ID ';'

codePart                                ::=          CODE ':' CADENA ';'
                                           |         CODE ':' LISTVALUE ';'

taskKnowledgeDefinitions                ::=          taskKnowledgeDefinition +

taskKnowledgeDefinition                 ::=          TASK-KNOWLEDGE ID ';'
                                                     [includePart]
                                                     [taskKnowledgeElements]
                                                     END TASK-KNOWLEDGE ID ';'

taskKnowledgeElements                   ::=          taskKnowledgeElement +

taskKnowledgeElement                    ::=          taskDefinition
                                           |         taskMethodDefinition

taskDefinition                          ::=          TASK ID ';'
                                                     GOAL ':' CADENA ';'
                                                     ROLES ':'
                                                      INPUT ':' taskRoleList
                                                      OUTPUT ':' taskRoleList
                                                     METHOD ':' ID ';'
                                                     END TASK ID ';'

taskRoleList                            ::=          taskRole +

taskRole                                ::=          ID ':' CADENA ';'

taskMethodDefinition                    ::=          TASK-METHOD ID ';'
                                                     REALIZES ':' ID ';'
                                                     [decompositionsPart]
                                                     ROLES ':'
                                                     INTERMEDIATE ':' intermediateRoleList
                                                     [codePart]
                                                     END TASK-METHOD ID ';'

intermediateRoleList                    ::=          intermediateRole +

intermediateRole                        ::=          ID ':' CADENA ';'

decompositionsPart                      ::=          DECOMPOSITION ':'
                                                      [INFERENCES ':' IDList ';']
                        [TASKS ':' IDList ';']
```

BIBLIOGRAFÍA

[Aitchison 75] J. Aitchison, I. R. Dunsmore. Statistical Prediction Analysis. Cambridge: Cambridge University Press, 1975

[Akaike 73] H. Akaike. Information Theory and an Extension of the Maximum Likelihood Principle. In Second International Symposium on Information Theory, eds B. N. Petrov, F. Cáski, pp. 267-281. Budapest: Akademiai Kaidó. 1973

[Akaike 74] H. Akaike. A New Look at Statistical Model Identification. IEEE Transactions on Automatic Control 19, pp. 716-723, 1974

[Alan 94] A. Stuart, K. Ord. Kendall's Advanced Theory of Statistics, 6th Edition, Volume I, Distribution Theory. Edward Arnold, 1994

[Arnold 95] B. C. Arnold, R. A. Groeneveld. Measuring Skewness with Respect to the Mode. The American Statistician, vol. 49, No. 1, 1995

[Bahadur 71] R. R. Bahadur. Some Limit Theorems in Statistics. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1971

[Berger 85] J. O. Berger. Statistical Decision Theory and Bayesian Analysis, 2nd Edition. Springer Series in Statistics, 1985

[Bernardo 00] J. M. Bernardo, A. F. Smith. Bayesian Theory. Wiley Series in Probability and Statistics, 2000

[Bishop 95] C. M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, 1995

[Boehm 88] B. Boehm. A Spiral Model of Software Development and Enhancement. Computer, pp. 61-72, 1988

[Booch 98] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modelling Language User Guide. Reading, M.A., Addison Wesley, 1998

[Borovkov 88] A. A. Borovkov. Estadística Matemática. Moscú, 1988

[Breiman 84] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. Classification and Regression Trees. Monterey, CA: Wadsworth and Brooks/Cole, 1984

[Bridle 90] J. S. Bridle. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In Neuro-Computing: Algorithms, Architectures and Applications. Eds. F. Fogleman Soulié and J. Hérault, pp. 227-236. Berlin: Springer, 1990

[Brown 96] B. M. Brown, T. P. Hettmansperger. Normal Scores, Normal Plots and Tests for Normality. Journal of the American Statistical Association, Vol. 91, No. 436, Theory and Methods, pp. 1668-1675, 1996

[Bryson 69] A. E. Bryson, Y. C. Ho. Applied Optimal Control. Blaisdel, 1969

[Campbell 80] N. A. Campbell. Shrunken Estimators in Discriminant and Canonical Variate Analysis. Applied Statistics 29, pp. 5-14, 1980

[Chandrasekaran 88] B. Chandrasekaran. Generic Tasks as Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples. The Knowledge Engineering Review, vol. 3, 3, pp. 183-210, 1988

[Chandrasekaran 90] B. Chandrasekaran. Design Problem Solving: A Task Analysis. AI Magazine, 1990

[Chandrasekaran 93] B. Chandrasekaran, T. R. Johnson. Generic Tasks and Task Structures: History, Critique and New Directions. In David, J. M., Krivine, J. P., and Simmons R., editors, Second Generation Expert Systems, Berlin, Springer Verlag, 1993

[Cover 91] T. M. Cover, J. A. Thomas. Elements of Information Theory. Wiley Series in Telecommunications, 1991

[Cox 74] D. R. Cox, D. V. Hinkley. Theoretical Statistics. Chapman and Hall, 1974

[D'Agostino 73] R. B. D'Agostino, E. S. Pearson. Testing for Departures from Normality. I. Fuller Empirical Results for the Distribution of b_2 and $\sqrt{b1}$. Biometrika, 60, pp. 613-622

[D'Agostino 90] R. B. D'Agostino, A. Belanger, R. B. D'Agostino (JR.). A Suggestion for Using Powerful and Informative Tests of Normality. The American Statistician, vol. 44, Nº 4, pp. 316-321, November 1990

[Dasarathy 91] B. V. Dasarathy. (ed.) Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. Los Alamitos, CA: IEEE Computer Society Press, 1991

[Dash 97] M. Dash, H. Liu. Feature Selection for Classification. Intelligent Data Analysis, Elsevier Science Inc, 1997

[Dembo 98] A. Dembo, O. Zeitouni. Large Deviations Techniques and Applications. Springer Verlag, 1998

[Devijver 82] P. A. Devijver, J. V. Kittler (eds). Pattern Recognition. A Statistical Approach. Englewood Cliffs, NJ: Prentice Hall, 1982

[Donelly 95] C. Donelly, R. Stallman. Bison: The YACC-Comaptible Parser Generator V. 1.25, 1995

[Doornik 94] J. A. Doornik, H. Hansen. A Practical Test for Univariate and Multivariate Normality. Discussion Paper, Nuffield College, 1994

[Duda 73] R. O. Duda, P. E. Hart. Pattern Clasification and Scene Analysis. J. Wiley, 1973

[Efron 82] B. Efron. The Jackknife, the Bootstrap and Other Resampling Plans. Philadelphia: SIAM, 1982

[Elderton 69] W. P. Elderton, N. Ll. Johnson. Systems of Frequency Curves. Cambridge University Press, 1969

[Fisher 36] R. A. Fisher, The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics, 7, pp. 179-188, 1936

[Friedman 89] J. H. Friedman. Regularized Discriminant Analysis. Journal of the American Statistical Association 84, pp. 165-175, 1989

[Fukunaga 90] K. Fukunaga. Introduction to Statistical Pattern Recognition, 2nd Edition. Academic Press, 1990

[Geisser 93] S. Geisser. Predictive Inference: An Introduction. New York: Chapman & Hall, 1993

[Geman 92] S. Geman, E. Bienenstock, R. Doursat. Neural Networks and the Bias/Variance Dilema. Neural Computation 4 (1), pp. 1-58, 1992

[Gharahmani 94] Z. Gharahmani, M. I. Jordan. Supervised Learning from Incomplete Data via an EM Approach. In J. D. Cowan, G. T. Tesauro, and J. Alspector (Eds.), Advances in Neural Information Processing Systems, Volume 6, pp. 120-127. San Mateo, CA: Morgan Kaufmann, 1994

[Giarratano 98] CLIPS Reference Manual: Volume II, Advanced Programming Guide. V. 6.10, 1998. Disponible en http://www.ghg.net/clips/clips.html

[Hampshire 90] J. B. Hampshire, B. Pearlmutter. Equivalence Proofs for Multilayer Perceptron Classifiers and the Bayesian Discriminant Function. Proceedings of the 1990 Connectionist Models Summer School, Morgan Kaufmann, pp. 159-172, 1990

[Hand 82] D. J. Hand. Kernel Discriminant Analysis. Chichester: Research Studies Press, 1982

[Hinton 86] G. E. Hinton, Learning Distributed Representations of Concepts. In Proceedings of the Eight Annual Conference of the Cognitive Science Society (Amherst, 1986), pp. 1-12, Hillsdale: Erlbaum, 1986

[Hoffbeck 96] J. P. Hoffbeck, D. A. Landgrebe. Covariance Matrix Estimation with Limited Training Data. IEEE Transactions on Pattern Analysis and Machine Intelligence, V. 18, 7, pp. 763-767, 1996

[Hopfield 87] J. J. Hopfield. Learning Algorithms and Probability Distributions in Feed-Forward and Feed-Back Networks. Proceedings of the National Academy of Sciences of the USA 84, pp. 8429-8433, 1987

[Huber 81] P. J. Huber. Robust Statistics. New York: Wiley, 1981

[Hugues 68] G. F. Hugues. On the Mean Accuracy of Statistical Pattern Pattern Recongnizers. IEEE Trans. Info. Theory, IT-14, pp. 55-63, 1968

[Hush 89] D. R. Hush. Classification with Neural Networks: A Performance Analysis. In Proceedings of the IEEE International Conference on Systems Engineering, pp. 277-280, 1989

[Jiménez 98] L. Jiménez, D. Landgrebe. Supervised Classification in High Dimensional Space: Geometrical, Statistical and Asymptotical Properties of Multivariate Data. IEEE Transactions on Systems, Man, and Cybernetics, 1998.

[Johnson 49] N. L. Johnson. Systems of Frequency Curves Generated by Methods of Translation. Biometrika, 36, 149, 1949

[Kullback 68] S. Kullback. Information Theory and Statistics. Dover Publications, 1968

[Lauritzen 88] S. Lauritzen, D. J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion). Journal of the Royal Statistical Society series B 50, pp. 157-224, 1988

[Lenat 90] B. D. Lenat, R. V. Guha. Building Large Knowledge-Based Systems. Reading, MA., Addison Wesley, 1990

[Levenberg 44] K. Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. Quarterly Journal of Applied Mathematics II (2), pp. 164-168, 1944

[Lin 80] C. C. Lin, G. S. Mudholkar. A Simple Test for Normality Against Asymmetric Alternatives. Biometrika 67, pp. 455-461, 1980

[Little 87] R. J. Little, D. B. Rubin. Statistical Analysis with Missing Data. New York: Wiley, 1987

[Mañas 94] S. Mañas, Diagnóstico de la Maduración Cerebral mediante Técnicas de Análisis Computerizado y Estadístico del EEG y Potenciales Evocados. Tesis Doctoral, Universidad de La Laguna, 1994

[Marcus 88] S. Marcus, editor. Automatic Knowledge Acquisition for Expert Systems. Boston, Kluwer, 1988

[Mardia 79] K. V. Mardia, J. T. Kent, J. M. Bibby. Multivariate Analysis. London: Academic Press, 1979

[Marquardt 63] D. W. Marquardt. An Algorithm for Least Squares Estimation of Non-Linear Parameters. Journal of the Society of Industrial and Applied Mathematics 11 (2), pp. 431-441MATLAB Neural Network Toolbox. 1996

[MATLAB 96] MATLAB Reference Guide, 1996

[McCarthy 60] J. McCarthy. Programs with Common Sense, Proceedings of the Teddington Conference on the Mechanization of Thought Processes. H. M. Stationary Office, pp. 77-84. 1960

[Moran 79] M. A. Moran, B. J. Murphy. A Closer Look at Two Alternative Methods of Statistical Discrimination. Applied Statistics 28, pp. 223-232, 1979

[Murphy 95] P. M. Murphy, D. W. Aha. UCI Repository of Machine Learning Databases [Machine-Readable Data Repository]. Irvine, CA: University of California, Dept. of Information and Computer Science. Available by anonymous ftp from ics.uci.edu in directory pub/machine-learning-databases

[Newell 82] A. Newell. The Knowledge Level. Artificial Intelligence, 18, pp. 82-127, 1982

[Nguyen 90] D. Nguyen, B. Widrow. Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights. International Joint Conference of Neural Networks, vol. 3pp. 21-26, 1990

[Paxson 95] V. Paxson. Flex: A Fast Scanner Generator V. 2.5, 1991

[Pearl 88] J. Pearl. Probabilistic Inference in Intelligent Systems. Networks of Plausible Inference. San Mateo, CA: Morgan Kaufmann, 1988

[Pearson 63] E. S. Pearson. Some Problems Arising in Approximating to Probability Distributions, Using Moments. Biometrika, 24, 404, 1963

[Piñeiro 96] J. D. Piñeiro. Tesis Doctoral: Hacia un Sistema Basado en el Conocimiento para el Diagnóstico de Patologías en Señales Cerebrales. 1996

[Press 92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. Numerical Recipes in C. The Art of Scientific Computing, 2nd Edition. Cambridge University Press, 1992

[Quenouille 49] M. H. Quenouille. Approximate Tests of Correlation in Time Series. Journal of the Royal Statistical Society series B 11, pp. 68-84MATLAB Neural Network Toolbox. 1996

[R 00] An Introduction to R, 2000. Disponible en http://cran.r-project.org

[Rich 91] E. Rich, K. Knight. Artificial Intelligence, 2nd Edition, McGraw-Hill, 1991

[Ripley 88] B. D. Ripley. Statistical Inference for Spatial Processes. Cambridge: Cambridge University Press, 1988

[Ripley 96] B. D. Ripley. Pattern Recognition and Neural Networks. Cambridge University Press, 1996

[Ripley. 94] B. D. Ripley. Neural Networks and Flexible Regression and Discrimination. In Statistics and Images 2, ed. K. V. Mardia. Advances in Applied Statistics 2, pp. 39-57. Abingdon: Carfax

[Romeu 93] J. L. Romeu, A. Ozturk. A Comparative Study of Goodness-of-Fit Tests for Multivariate Normality. Journal of Multivariate Analysis 46, pp. 309-334, 1993

[Rosenblatt 62] F. Rosenblatt. Principles of Neurodynamics. Washintong, DC: Spartan Books, 1962

[Rousseeuw 90] P.J. Rousseeuw, B. C. Van Zomeren. Unmasking Multivariate Outliers and Leverage Points (with discussion). Journal of the American Statistical Association 85, pp. 633-651, 1990

[Royston 82] J. P. Royston. An Extension of Shapiro and Wilk's W Test for Normality to Large Samples. Applied Statistics, 31, pp. 115-124, 1982

[Rumelhart 86] D. E.. Rumelhart, J. L. McClelland. PDP Research Group. Parallel Distributed Processing. Volume 1: Foundations. MIT Press, 1986

[Russell 95] S. J. Russell, P. Norvig. Artificial Intelligence. Prentice Hall, 1995

[Sánchez 93] J.L. Sánchez. Métodos para el Procesamiento y Análisis Estadístico Multivariante de Señales Multicanal. Tesis Doctoral, Universidad de La Laguna, 1993

[Schreiber 93] G. Schreiber, B. Wielinga, J. Breuker. KADS A Principled Approach to Knowledge-Based System Development. Academic Press, 1993

[Schreiber 94] G. Schreiber, B. Wielinga, R. De Hoog. CommonKADS: A Comprehensive Methodology for KBS Development. IEEE Expert, pp. 28-36, December 1994

[Schreiber 99] G. Schreiber. Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press, 1999

[Shapiro 65] S. S. Shapiro, M. B. Wilk. An Analysis of Variance Test for Normality (Complete Samples). Biometrika 52, pp. 591-611, 1965

[Shapiro 72] S. S. Shapiro, R. S. Francia. An Approximate Analysis of Variance Test for Normality. Journal of the American Statistical Association, 67, pp. 215-216, 1972

[Sommerville 95] I. Sommerville. Software Engineering. Harlow, U.K., Addison Wesley, 1995

[Steels 90] L. Steels. Components of Expertise. AI Magazine, summer, 1990

[Toolbox-Neur. 92] Neural Network Toolbox for Use with MATLAB, 1992

[Toolbox-Neur. 96] MATLAB Neural Network Toolbox. 1996

[Toolbox-Stats. 92] Statistics Toolbox for Use with MATLAB, 1992

[Toolbox-Stats. 96] MATLAB Statistics Toolbox. 1996

[Toro 80] J. Toro, M. Cervera. Test de Análisis de Lectoescritura [Reading and Spelling Test] Madrid: Aprendizaje Visor, 1980

[Tu 95] S. W. Tu, H. Eriksson, J. H. Gennari, Y. Shahar, M. A. Musen. Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge Acquisition Tools: The Application of PROTÉGÉ II to Protocol-Based Decision Support. Artificial Intelligence in Medicine, 7(5), 1995

[Vapnik 82] V. N. Vapnik. Estimation of Dependencies Based on Empirical Data. New York: Springer, 1982

[Vasicek 76] O. Vasicek. A Test for Normality Based on Sample Entropy. Journal of the Royal Statistics Society. B, 38, pp. 54-59

[Venables 99] W. N. Venables, B. D. Ripley. Modern Applied Statistics with S-Plus, 3rd Edition. Springer Verlag, 1999

[Watanabe 69] S. Watanabe. Knowing and Guessing. New York: Wiley, 1969

[Weiss 84] S. M. Weiss, C. A. Kulikowski. A Practical Guide to Designing Expert Systems. Rowman & Allanheld, 1984

[Wielinga 92] B.J. Wielinga, A. T. Schreiber, J. A. Breuker. KADS: A Modelling Approach to Knowledge Engineering. Knowledge Acquisition, 4(1), Special Issue 'The KADS Approach to Knowledge Engineering', 1992