UNED

Escuela Técnica Superior de Ingeniería Informática

Máster en Inteligencia Artificial Avanzada:
Fundamentos, Métodos y Aplicaciones

Especialidad en Sistemas Inteligentes
de Diagnóstico, Planificación y Control

Aprendizaje de redes bayesianas en Carmen

Jesús Oliva Gonzalo

Tutores:
Prof. Manuel Arias Calleja
Prof. Francisco Javier Díez Vegas

Madrid - Septiembre de 2008

Índice general

Índice de figuras

Agradecimientos

A todas las personas que me han ayudado en este trabajo. En especial, a mis tutores, los Profs. Manuel Arias y Francisco Javier Díez, por su ayuda, consejos y colaboración y por haberme dado autorización para incluir en esta memoria fragmentos de su tesis doctoral, el primero, y de sus apuntes sobre modelos gráficos probabilistas, el segundo.

También me gustaría darle las gracias a D. Agustín Gómez de la Cámara, Jefe de la Unidad de Investigación del Hospital Universitario 12 de Octubre de Madrid, por haber puesto a mi disposición la base de datos sobre enfermos de cáncer de pulmón utilizada en este trabajo y por las discusiones mantenidas que han servido como realimentación para orientar las funcionalidades incluidas en el módulo desarrollado. Además, he de agradecer la ayuda económica del programa de financiación de acciones específicas de CIBERESP –Centro de Investigación Biomédica en Red de Epidemiología y Salud Pública– y al Grupo Colaborativo para el Estudio del Carcinoma Broncogénico del Servicio de Neumología del Hospital 12 de Octubre.

Además, me gustaría agradecer a D. José Javier Trujillano Cabello, por facilitarme la base de datos sobre pacientes ingresados en la UCI polivalente del Hospital Universitario Arnau de Vilanova de Lleida y a D. Antonio Ríos Zamarro, D. Manuel Padial y al resto de integrantes del Área de Control Integral del Riesgo de CajaMadrid por haberme facilitado la base de datos de riesgo en hipotecas y ayudarme a interpretar sus variables.

A mi familia, por su incondicional apoyo en todo lo que me he propuesto y por esa mezcla de libertad y responsabilidad que siempre me han inculcado y que me ha llevado a conseguir todas mis metas.

A mis amigos, que siempre han aportado esa pizca de alegría y locura necesaria para seguir adelante.

De corazón, Gracias.

Prólogo

Es un hecho destacable que una ciencia que empezó analizando juegos de azar acabe convirtiéndose en el más importante objeto del conocimiento humano.

Pierre Simon de Laplace, "Théorie Analytique des Probabilités".

Cada día, miles de personas en todo el mundo apuestan su dinero en distintos juegos de azar, a sabiendas de que las posibilidades de conseguir el premio son realmente escasas. Tanto es así que, sólo en España, la cantidad jugada en 2005 ascendió a 28.334 millones de euros. En el estado de Nevada, los juegos de azar salvaron a una pequeña ciudad llamada Las Vegas de un futuro más que dudoso, convirtiéndola en una "Meca" a la que peregrinan treinta millones de personas cada año con el único objetivo de apostar su dinero en casinos que, gracias a la comprensión de las estructuras del azar, se aseguran un promedio de tres céntimos de cada dolar apostado, generando así unos beneficios anuales de 16.000 millones de dólares. Pero la cosa no queda ahí, muchos individuos que proclaman su oposición a los juegos de azar realizan apuestas no obstante, basándose en el valor que otorgan a sus vidas, sus casas, sus coches y otras pertenencias, puesto que eso es exactamente lo que hacemos cuando contratamos pólizas de seguros (de hecho, hasta bien entrado el siglo XVIII, la emisión de seguros de vida era ilegal en todos los países europeos, a excepción de Inglaterra). Como dice Ian Stewart en su libro "De aquí al infinito"[36]:

[...] el cálculo de probabilidades, y su instrumento de aplicación, la estadística, han mantenido a menudo unas relaciones más bien difíciles con el juego. Por ejemplo, lo que sucede con los seguros de vida: uno apuesta a que se va a morir y la compañía de seguros a que no. Para ganar, tenemos que perder.

A la vista de las ingentes cantidades de dinero que mueven y la gran cantidad de sitios en donde encontramos la influencia de las probabilidades, parece mentira que el origen de la teoría matemática que permite manejarlas resida en la correspondencia que, a mediados del siglo XVII, mantuvieron dos franceses: Blaise Pascal y Pierre de Fermat.

Desde que, en 1525, Girolamo Cardano, en su obra *Libro sobre los juegos de azar*, diera el primer paso hacia una teoría del azar, hasta que, en 1933, el matemático soviético Andréi

Kolmogórov propuso un sistema de axiomas para la teoría de la probabilidad, basado en la teoría de conjuntos y en la teoría de la medida, desarrollada pocos años antes por Lebesgue, Borel y Frechet entre otros, han sido muchos los matemáticos que se han preocupado por el conocimiento de las estructuras abstractas del azar. Así, durante décadas, la teoría de la probabilidad ha ido evolucionando, hasta dar lugar a una fértil área que fija sus raíces en la teoría de la medida y encuentra aplicación en las más variadas ramas del conocimiento, como puede ser la física (donde corresponde mencionar el desarrollo de las difusiones y el movimiento Browniano), o las finanzas (donde destaca el modelo de Black y Scholes para la valuación de acciones).

Sin embargo, a pesar de los esfuerzos de los Pascal, Fermat, Bernoulli (Jacob, Daniel y Nicolas), Gauss o el propio Thomas Bayes, el ser humano presenta una gran carencia en cuanto a la estimación subjetiva de la probabilidad. Cientos de miles de años de evolución han servido para desarrollar capacidades mentales realmente impresionantes, pero cuando se trata de manejar las probabilidades, el ser humano se equivoca sistemáticamente. Existen numerosos ejemplos que permiten comprobar este tipo de errores, como el conocido "caso de los taxis" o la "falacia del fiscal":

Supongamos que Londres, la ciudad natal de Bayes, tiene dos compañías de taxis: Taxis Azules, con una flota de 15 taxis y Taxis Verdes, con una flota de 85 taxis. Una noche uno de los taxis atropella a un peatón en presencia de un testigo que, posteriormente, declara que se trataba de un taxi azul. Para comprobar la fiabilidad del testimonio, se realizan varias pruebas en condiciones similares a las del accidente y se comprueba que el testigo es capaz de identificar correctamente el color del taxi en un 80 % de las veces. Seguramente, a la vista de un testimonio tan fiable, la mayoría de nosotros condenaría a la compañia Azul, ya que resulta fácil pensar que las probabilidades de que el culpable haya sido un taxi azul son del 80 %. Sin embargo, si el reverendo Bayes levantara la cabeza señalaría a la compañía Verde como culpable del atropello. Lo que nuestro sentido común ignora en muchos casos, pero el método de Bayes considera adecuadamente, es que existe una probabilidad de 0,85 de que un taxi *cualquiera* de la ciudad sea verde. Esto es lo que llamamos *probabilidad a priori*. Sin el testimonio del testigo, la probabilidad de que el culpable sea un taxi verde es del 85 % y, tras obtener la información del testigo, no debemos olvidar este dato. En lugar de ello, la fiabilidad de la evidencia del testigo, debe combinarse con la probabilidad a priori para obtener la probabilidad real y el método de Bayes nos indica cómo hacerlo:

Si denotamos como 'A' el suceso "el taxi es azul" (análogamente "el taxi es verde" para calcular la probabilidad de que el atropello lo haya cometido un taxi verde) y como 'B' el suceso "el testigo ha dicho que el taxi es azul", la probabilidad real que debemos calcular viene dada por la conocida formula:

$$P(\mathrm{A} \mid \mathrm{B}) = \frac{P(\mathrm{B} \mid \mathrm{A}) \times P(\mathrm{A})}{P(\mathrm{B})} \tag{1}$$

Así, con unos sencillos cálculos podemos comprobar que la probabilidad real de que

el taxi que ha causado el accidente sea de la compañía Azul es tan sólo de un 41 %. A la vista de este resultado, nuestro veredicto cambiaría radicalmente. Este ejemplo permite comprobar que la tendencia del ser humano a obviar las probabilidades a priori a la hora de realizar una estimación, puede llevar a calificar como "casi seguro" un hecho que realmente no es el más probable, con los riesgos que ello conlleva.

Otro de los errores de interpretación que tiende a cometer el ser humano consiste en confundir una probabilidad condicionada con su "opuesta" (es decir, considerar que $P(A|B) = P(B|A)$)[1]. Este error de interpretación es más corriente de lo deseable, hasta el punto de ser conocido en estadística forense como "la falacia del fiscal": Supongamos que en una ciudad de 10 millones de habitantes se ha cometido un crimen. Junto al cadáver se encontraron restos de ADN del asesino que fueron comparados con los de un sospechoso, comprobándose que las marcas de ADN coincidían. Los laboratorios que realizaron el análisis afirman que éste tiene una fiabilidad del 99.999999 % (es decir, la probabilidad de que el análisis sea erróneo es de 1 entre 1 millón). En estas condiciones, el fiscal del caso se frota las manos: "Dado que el ADN del acusado coincide con el encontrado junto al cadáver, la probabilidad de que éste sea inocente es de 1 entre 1 millón". Ante una probabilidad tan apabullante, cualquiera de nosotros condenaría al acusado, sin embargo, a falta de otras pruebas, el reverendo Bayes lo declararía inocente sin lugar a dudas. Efectivamente, en el argumento del fiscal hay un error sutil, pero cuyas consecuencias son devastadoras. Llamemos A al suceso "el acusado es inocente"[2] y B al suceso "las marcas de ADN coinciden". Entonces, lo que nos ha dicho el fiscal es que $P(A|B) = 0{,}000001$ pero esto es radicalmente falso!!! Lo que nos aporta el análisis de ADN es que $P(B|A) = 0{,}000001$, es decir, la probabilidad de que las marcas de ADN coincidan teniendo en cuenta que el acusado es realmente inocente y no la probabilidad de que sea inocente dado que las marcas coinciden. Para calcular la verdadera probabilidad debemos recurrir nuevamente al Teorema de Bayes:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} = \frac{\frac{1}{1{,}000{,}000} \times 0{,}9999999}{P(B)}$$

$$P(\neg A|B) = \frac{P(B|\neg A) \times P(\neg A)}{P(B)} = \frac{0{,}999999 \times \frac{1}{10{,}000000}}{P(B)}$$

Dividiendo ambas formulas tenemos que[3]:

$$\frac{P(A|B)}{P(\neg A|B)} \approx 10^{-6} \times 10^{7} = 10$$

[1] Obviamente, una probabilidad condicionada no tiene por qué coincidir con su opuesta. Supongamos que A representa el suceso "tener cuatro patas" y B "ser un perro", entonces $P(A|B) = 1$ pero $P(B|A)$ está muy por debajo de ese valor, ya que hay muchos más animales que tienen 4 patas.

[2] Denotaremos por $P(\neg A)$ la probabilidad de que sea culpable.

[3] Véase que la probabilidad a priori de que el acusado sea inocente (P(A)) es de 0.9999999 puesto que, entre los 10 millones de habitantes de la ciudad hay un culpable y 9.999.999 inocentes.

Por tanto, la probabilidad de que el acusado sea inocente es diez veces mayor que la probabilidad de que sea culpable. Nuestro sentido común nos estaba llevando a condenar a una persona que tiene aproximadamente el 90 % de probabilidades de ser inocente!!! Alguien puede pensar que este ejemplo es un mero truco rebuscado, sin embargo, en la literatura forense pueden encontrarse casos de sentencias anuladas por estar dictadas tomando en consideración la "falacia del fiscal". Uno de ellos, el caso de un asalto con robo a una mujer ocurrido en 1964 en Los Ángeles ("El Pueblo contra Collins", 68 Cal. 2d 319, 438 P. 2d 33, 66 Cal. Rptr. 497), en el que se anuló la sentencia aún cuando la probabilidad involucrada era de 1 entre 12 millones.

Estos y otros muchos errores son habituales al tratar con probabilidades. Por tanto, dado que el proceso de evolución natural no ha dotado al ser humano de una gran habilidad para estimar y manejar probabilidades, parece, cuando menos, conveniente hacer un uso sistemático de las herramientas matemáticas desarrolladas para manejar las probabilidades y así, obtener unos resultados más fiables. En este ámbito destacan las redes bayesianas que constituyen la base del trabajo que se expone a continuación.

Capítulo 1

Introducción

Quiero decir, para empezar: ¿a quién le importa si saco una bola blanca o una bola negra de una urna? Y segundo: si tan preocupado estás por el color de la bola que sacas, no lo dejes en manos del azar: ¡Mira en la maldita urna y saca la bola del color que quieras!

Stephanie Plum (personaje de ficción creado por Janet Evanovich).

1.1. Presentación

El *aprendizaje* puede ser definido como "cualquier proceso a través del cual un sistema mejora su comportamiento" [16]. La habilidad de aprender es una de las características centrales de los *sistemas inteligentes*, y por esa razón, en la investigación y desarrollo de este área se ha invertido un gran esfuerzo y dedicación. En este sentido, la automatización del proceso de adquisición de conocimientos se considera uno de los problemas principales en la construcción de estos sistemas. Un aspecto importante del aprendizaje inductivo es el de la obtención de modelos que representen un dominio de conocimiento. En particular, en los sistemas donde se desea predecir el valor de algunas variables a partir de otras, es muy importante obtener las relaciones de dependencia existentes entre las distintas variables. Una representación del conocimiento que es capaz de capturar esta información sobre las dependencias entre las variables son las redes bayesianas [30, 16].

Las *redes bayesianas* son modelos probabilísticos multivariados que relacionan un conjunto de variables aleatorias mediante un grafo dirigido acíclico en el que cada nodo es una variable y cada enlace una dependencia probabilística. Las mismas proveen una forma compacta de representar el conocimiento y métodos flexibles de razonamiento —basados en la teoría de la probabilidad— capaces de predecir el valor de variables no observadas y explicar las observadas. La obtención de una red bayesiana a partir de datos es un proceso de aprendizaje que se divide en dos etapas: el aprendizaje estructural y el aprendizaje parámetrico [28, 27]. La primera de ellas consiste en obtener la estructura de la red bayesiana, es decir, las relaciones de dependencia e independencia entre las variables involucradas. La segunda etapa, tiene como finalidad obtener las probabilidades a priori

y condicionales requeridas, a partir de una estructura dada.

Este trabajo de fin de máster trata del diseño e implementación de un módulo de *aprendizaje automático* de *redes bayesianas* para Carmen. Carmen es una herramienta multiplataforma de código libre para el manejo de modelos gráficos probabilistas que nace con la intención de ofrecer a la comunidad científica un sistema que pueda ser utilizado por distintos grupos de investigación para implementar y probar nuevos algoritmos así como para construir nuevas aplicaciones. El punto de partida del proyecto Carmen es el proyecto Elvira, un proyecto financiado por la CICYT que se desarrolló entre los años 1997 y 2000. En él participaron 25 profesores de 8 universidades españolas, agrupados en cuatro subproyectos: Granada, Almería, País Vasco y UNED. El principal objetivo del proyecto era la construcción de un entorno que sirviera, por un lado, para la investigación de nuevos métodos y algoritmos de razonamiento probabilístico y, por otro, para la implementación de sistemas expertos bayesianos. A finales de 2001, el Ministerio de Ciencia y Tecnología concedió el Proyecto Coordinado *Elvira II: Aplicaciones de los Modelos Gráficos Probabilísticos*, cuyos objetivos eran dos: mejorar las características del programa Elvira y desarrollar aplicaciones en diversos campos como la medicina, la genética, la agricultura y el comercio inteligente (filtrado cooperativo).

El principal problema de Elvira es que en su desarrollo no se ha seguido una metodología basada en la ingeniería del software. Del mismo modo, durante los últimos años se han desarrollado varias herramientas para el manejo de modelos gráficos probabilistas tanto comerciales como de código abierto. Sin embargo, ninguna de ellas ha conseguido imponerse, quizás debido a la dificultad de mantenimiento y falta de fiabilidad provocadas por un diseño y desarrollo alejado de los principios básicos de la ingeniería del software. En este sentido, la herramienta Carmen en general, y el módulo de aprendizaje en particular, se han diseñado siguiendo estos principios y manteniendo una serie de requisitos como son la robustez y eficiencia necesarias para formar parte de otras aplicaciones, la escalabilidad de los distintos algoritmos para permitir que trabajen con redes de cientos o miles de nodos, o la extensibilidad.

En este contexto se desarrolla el módulo de aprendizaje presentado en este trabajo. Para adecuarse a los requerimentos de Carmen se ha llevado a cabo un cuidado proceso de diseño de los algoritmos de aprendizaje, que constituyen el núcleo de éste módulo, precedido por un estudio del estado de la técnica. En la etapa de diseño se ha prestado especial atención al requisito de la extensibilidad, que es probablemente el más importante de ellos. De este modo, se ha generado una estructura de clases y paquetes que permite ampliar fácilmente las funcionalidades ofrecidas por este módulo, acompañada de una amplia documentación tanto interna (en forma de comentarios explicativos en el código unidos a las páginas HTML generadas por la herramienta Javadoc de Sun[1]) como externa, como el capítulo 3 de este trabajo.

En resumen, en este trabajo de fin de máster se presenta el desarrollo de un módulo de aprendizaje de redes bayesianas para la herramienta Carmen, diseñado e implementado

[1] `http://java.sun.com/j2se/javadoc/`

atendiendo a los fundamentos y principios de la ingeniería del software.

1.2. Objetivos

El objetivo principal de este trabajo de fin de máster es el desarrollo de un módulo de aprendizaje para la herramienta de manejo de modelos gráficos probabilistas Carmen. El objetivo de Carmen es implantarse entre la comunidad científica como una herramienta para la implementación y prueba de nuevos algoritmos así como para el desarrollo de nuevas aplicaciones. De este modo, se busca un módulo de aprendizaje con la robustez y eficiencia necesarias para formar parte de otras aplicaciones teniendo siempre en cuenta uno de los requisitos principales: la extensibilidad del mismo, es decir, el módulo diseñado ha de ser fácilmente ampliable por terceras personas ajenas al proyecto. Para ello, el diseño e implementación de los algoritmos de aprendizaje (que constituyen el núcleo de este trabajo) se ha de llevar a cabo siguiendo rigurosamente una metodología basada en los fundamentos de la ingeniería del software.

Como objetivos secundarios se pretende realizar una revisión del estado de la técnica en aprendizaje de redes bayesianas que permita sentar las bases teóricas para el desarrollo posterior del módulo de aprendizaje y por último se pretende estudiar el comportamiento del módulo desarrollado en distintos ámbitos, realizando el aprendizaje a partir de bases de datos reales, con el objetivo de determinar sus principales cualidades y limitaciones, así como las posibles líneas de trabajo futuro.

En resumen, este trabajo de fin de máster se ha propuesto tres objetivos fundamentales:

- El primero de los objetivos es de caracter teórico y consiste en revisar el estado de la técnica en aprendizaje en redes bayesianas, analizando las cuestiones generales del problema para luego pasar a un análisis más detallado de las técnicas de aprendizaje paramétrico y estructural más utilizadas.
- El segundo objetivo es de naturaleza práctica: el diseño e implementación del módulo de aprendizaje de la herramienta Carmen atendiendo a los fundamentos de la ingeniería del software para conseguir un producto eficaz, robusto y fácilmente ampliable.
- Por último, un tercer objetivo es la aplicación de dicho módulo a problemas de distintos ámbitos como el sanitario o el económico para mostrar sus capacidades y analizar las ventajas e inconvenientes de las redes bayesianas en cada uno de estos ámbitos.

1.3. Desarrollo y metodología

Para construir el módulo de aprendizaje de la herramienta Carmen, se han tenido muy en cuenta los fundamentos de la ingeniería del software, siguiéndose las etapas clásicas de análisis de requisitos, diseño, implementación y validación. Además, como se indica en [2], en el diseño de Carmen en general y de este módulo de aprendizaje en particular, se han seguido una serie de principios (véanse [15, 32] para una descripción más detallada):

- Consistencia: "Usar los mismos criterios durante todo el proceso de desarrollo".
- Simplicidad: "Hacer las cosas comunes, fáciles y las cosas raras, posibles". El código más simple es más fácil de entender, modificar y, por consiguiente, más robusto.
- Eliminación: Muy relacionado con el anterior. "Un diseño está terminado cuando no se le puede quitar nada más".
- Diseñar desde el contexto: El contexto ha de ser analizado primero, con el objetivo de tener una visión general del problema, antes de entrar en detalles.
- Diseñar para el cambio: Esto es una consecuencia del requisito de extensibilidad. Dado que es imposible saber de antemano las características que se necesitarán en un futuro, el sistema ha de estar preparado para ser ampliado sin modificar el código existente o modificarlo tan sólo con cambios menores.
- Encapsulamiento: Una clase no debe de depender de cómo estén implementadas otras clases, ni de qué clases derivan de ella, etc.
- Alta cohesión y bajo acoplamiento: Una alta cohesión implica que los contenidos de un objeto están estrechamente relacionados funcionalmente entre sí. Un bajo acoplamiento implica una baja interconexión entre las clases.
- Principio de Liskov: Una clase que deriva de otra debe mantener el comportamiento de la clase de la que deriva.

Además, en la etapa de diseño se ha aplicado también el enfoque CVA (Commonality and Variability Analysis), que consiste en analizar en primer lugar los puntos en común que comparten las distintas entidades (por ejemplo, las métricas basadas en entropía) con el objetivo de crear abstracciones (una clase abstracta para englobar todas esas métricas) y después identificar las diferencias y encapsularlas en subclases [32].

La aplicación de estos principios se facilita mediante el uso de patrones de diseño [15, 17]. Un patrón es una solución ya probada a un problema que ocurre de un modo recurrente, y puede ser aplicado en varios contextos. De acuerdo con Shalloway y Trott [32], los patrones han de ser entendidos como guías para el análisis de problemas más que como recetas cerradas:

> "Si entiendes los principios de los patrones y trabajas en un proyecto donde un patrón desconocido para ti se puede aplicar, entonces seguramente extraigas el patrón por ti mismo."

1.4. Organización de la memoria

Esta memoria está compuesta por cinco capítulos cuya organización se basa en el esquema de: introducción, revisión de los aspectos teóricos, análisis, diseño e implementación, aplicación de los algoritmos implementados y conclusiones del trabajo.

El primero de los capítulos es una introducción, que engloba la presente sección y trata de recoger los objetivos del Trabajo de Fin de Máster realizado, así como la metodología utilizada durante el desarrollo del mismo.

El segundo capítulo consiste en una revisión del estado de la técnica en aprendizaje de redes bayesianas, centrándose en las técnicas de aprendizaje estructural mediante búsqueda heurística. En primer lugar, en las dos primeras secciones se hace un análisis general del problema del aprendizaje automático, planteando las cuestiones de carácter general de un modo introducctorio. En las siguientes secciones se detallan en profundidad las técnicas de aprendizaje paramétrico y aprendizaje estructural más utilizadas. Es importante tener en cuenta que para entender algunos de los conceptos, técnicas y algoritmos explicados en este capítulo, es necesario tener unas nociones básicas sobre teoría de la probabilidad, teoría de grafos y redes probabilistas.

El tercer capítulo muestra las etapas de diseño e implementación del módulo de aprendizaje presentado en este trabajo, haciendo especial hincapié en la pieza clave de dicho módulo: los algoritmos de aprendizaje. En la sección 3.1 se muestran paso a paso la fase de análisis de requisitos y la fase de diseño (distinguiendo entre el diseño arquitectónico y el diseño detallado) haciendo uso de diagramas de clases y de secuencias que permiten ver tanto la estructura (punto de vista estático) como el comportamiento (punto de vista dinámico) de un algoritmo genérico. En la sección 3.2 todo lo indicado en el punto anterior se muestra mediante un ejemplo concreto: el *algoritmo del gradiente*. A continuación, en la sección 3.3, se recoge, de forma análoga, el análisis y diseño de una métrica genérica. En la sección 3.4 se explica la estructura arbórea utilizada para almacenar los ejemplos de la base de datos de una forma eficiente, minimizando el espacio ocupado por dichos ejemplos en memoria y permitiendo el uso de algoritmos eficientes para el manejo de dichos datos. En concreto, se explica en detalle el algoritmo utilizado para calcular la frecuencia absoluta de cada configuración de una serie de variables. Por último, en la sección 3.5 se comentan las cuestiones relativas al diseño e implementación de la interfaz gráfica del módulo de aprendizaje.

Este capítulo tiene un doble objetivo: por una parte, se pretende mostrar que en el desarrollo del módulo de aprendizaje, se han utilizado los fundamentos de la ingeniería del software, con la intención de desarrollar un producto eficiente y robusto. Por otra parte, esta sección puede servir a otros programadores que pretendan ampliar las capacidades del módulo de aprendizaje (como veremos más adelante, en el análisis de requisitos, uno de los requisitos principales es que se facilite en la medida de lo posible la ampliación de este módulo de aprendizaje mediante contribuciones externas).

En el cuarto capítulo se muestran dos de los posibles ámbitos de aplicación del módulo de aprendizaje implementado. En el ámbito de la medicina, se han aprendido redes a partir de una base de datos de enfermos de cáncer del Hospital Universitario 12 de Octubre de Madrid y una base de datos de pacientes ingresados en la UCI polivalente del Hospital Universitario Arnau de Vilanova de Lleida. En el ámbito económico, se han obtenido redes a partir de una base de datos de Caja Madrid, con información sobre deudas de hipotecas.

Por último, en el quinto capítulo se recogen las posibles líneas de trabajo futuro y las conclusiones de este trabajo, centrándose sobre todo en la consecución de los objetivos indicados en la sección 1.2, y en la valoración de las cualidades y limitaciones del módulo de aprendizaje desarrollado.

Además, se incluye un apéndice con el manual de usuario que, en un futuro, será distribuido con la herramienta Carmen.

Capítulo 2

Estado de la técnica

¿Cómo osamos hablar de leyes del azar? ¿No es, acaso, el azar la antítesis de cualquier ley?

Joseph Bertrand, "Calcul des Probabilités".

En este apartado se recoge el estado de la técnica de aprendizaje automático de redes bayesianas. Las dos primeras secciones son de carácter introductorio y pretenden dar una visión general del problema del aprendizaje de redes bayesianas así como mostrar una serie de conceptos básicos que serán utilizados en el resto del capítulo. A continuación, en la sección 2.3 se trata el tema del aprendizaje paramétrico para, en las secciones 2.4 y 2.5, abordar el tema del aprendizaje estructural a partir de relaciones de independencia y mediante búsqueda heurística respectivamente. Por último, en la sección 2.6, se comentan algunas cuestiones importantes relativas al aprendizaje de redes bayesianas, dando una serie de referencias que permiten al lector interesado profundizar en cada una de ellas. Es importante señalar que para entender correctamente los contenidos de este capítulo es necesario estar familiarizado con algunos conceptos básicos sobre probabilidad, teoría de grafos y redes bayesianas. El núcleo de este capítulo ha sido tomado de los apuntes del Profesor Francisco Javier Díez sobre modelos gráficos probabilistas [13], habiéndose introducido algunas modificaciones y ampliaciones para actualizar su contenido y adecuarlo a los objetivos de este capítulo.

Con la generalización del uso de los ordenadores, que se produjo sobre todo en la década de los 80, la disponibilidad de bases de datos no ha dejado de crecer exponencialmente. Ello hace que exista un interés cada vez mayor en extraer conocimiento de ellas, es decir, en construir modelos matemáticos que permitan predecir el futuro y tomar las mejores decisiones. Es lo que se conoce como *minería de datos*.

En muchas situaciones prácticas, no existe la posibilidad de construir la estructura de la red ni obtener las distribuciones condicionales de probabilidad a partir de un experto. Además, diferentes expertos pueden dar diferentes valores, a veces contradictorios, debido al carácter subjetivo del proceso. En estas situaciones, la estructura de dependencia y sus

distribuciones condicionales asociadas pueden estimarse a partir de un conjunto de datos. El cometido de este capítulo es dar una visión general de las técnicas de aprendizaje automático de redes bayesianas. Éste es un campo de gran actualidad, ya que podríamos decir que al menos un tercio de las publicaciones que se producen cada año sobre redes bayesianas están dedicadas a esta cuestión. Dado que se trata de un problema matemáticamente complejo y que la literatura existente es amplísima, nos vamos a limitar a explicar algunos conceptos y algoritmos básicos y a dar referencias para que el lector interesado pueda profundizar en el tema.

2.1. Planteamiento del problema

De forma general, podemos decir que el problema del aprendizaje consiste en construir, a partir de un conjunto de datos, el modelo que mejor represente la realidad, o mejor dicho, una porción del mundo real en la cual estamos interesados. Como en el caso de la construcción manual de redes bayesianas, el aprendizaje de este tipo de modelos tiene dos aspectos: el aprendizaje paramétrico y el aprendizaje estructural. En algún caso puede ocurrir que tengamos ya el grafo de la red, construido con la ayuda de un experto, y estemos interesados solamente en la obtención de las probabilidades condicionadas; sería un caso de aprendizaje paramétrico. Sin embargo, es más habitual que deseemos construir a partir de los datos la red completa, es decir, tanto el grafo como los parámetros de la red.

Al tratar el tema de aprendizaje en redes bayesianas, es muy importante tener presente que distintos grafos pueden ser probabilísticamente equivalentes, es decir, pueden representar las mismas relaciones de dependencia e independencia y/o las mismas distribuciones conjuntas para sus variables. Por ello, haciendo uso de la noción de independencia, o equivalencia de distribuciones, se puede obtener una división en clases de equivalencia del conjunto de todos los grafos dirigidos posibles de n variables.

Hay dos métodos principales para construir la red. El primero consiste en realizar, a partir de las frecuencias observadas en la base de datos, una estimación de la distribución de probabilidad que rige el mundo real; las relaciones de dependencia e independencia probabilista de dicha distribución indican cuál debe ser la estructura del grafo. Es decir, se trata de buscar un grafo que sea mapa de independencias de la distribución de probabilidad. Naturalmente, puede haber más de una solución, pues como hemos comentado existen grafos equivalentes en sentido probabilista, es decir, grafos que representan las mismas relaciones de independencias y de (posibles) dependencias.

El otro método de aprendizaje estructural consiste en realizar una búsqueda heurística utilizando alguna medida de calidad: en general, se parte de una red sin enlaces y se van añadiendo enlaces uno a uno hasta que la red representa adecuadamente la distribución de probabilidad obtenida de la base de datos. También en este método hay que tener en cuenta la existencia de grafos equivalentes en sentido probabilista.

Antes de ver con detalle cada uno de estos métodos (el aprendizaje paramétrico en

la sec. 2.3, el estructural basado en relaciones de independencia en la 2.4 y el estructural mediante búsqueda heurística en la 2.5), vamos a estudiar primero algunas cuestiones generales que nos ayudarán a entender mejor el aprendizaje de redes bayesianas.

2.2. Cuestiones generales sobre aprendizaje

2.2.1. El problema del sobreajuste

Hemos dicho antes que el problema del aprendizaje consiste en construir el modelo que mejor represente una porción del mundo real en la cual estamos interesados. Sin embargo, en la práctica no conocemos toda la realidad, sino sólo un conjunto de datos. Por ejemplo, si queremos construir un modelo para el diagnóstico de cáncer de hígado, no conocemos la realidad completa (todos los posibles pacientes), sino sólo los casos recogidos en cierta base de datos. Podríamos pensar entonces que el objetivo es construir el modelo que más se ajusta a los datos disponibles. Sin embargo, se comprueba en muchos casos que el modelo que mejor se ajusta *a los datos* no es necesariamente el que mejor se ajusta *a la realidad*.

Este fenómeno se denomina *sobreajuste* (en inglés, *overfitting*) y tiende a ocurrir cualquiera que sea el tipo de modelo que queremos aprender (ya sea un árbol de clasificación, una red neuronal, un conjunto de reglas difusas...). Veamos un par de ejemplos para entenderlo mejor:

Supongamos que queremos construir un modelo para el diagnóstico diferencial de tres enfermedades, E_a, E_b y E_c, a partir de una serie de hallazgos. Entre ellos se encuentra un síntoma S tal que $P(+s|e_a) = 0'40$, $P(+s|e_b) = 0'05$ y $P(+s|e_c) = 0'01$. Si la base de datos contiene 150 pacientes con la enfermedad E_c es posible que ninguno de ellos presente el síntoma S. (La probabilidad de que ocurra esto es $0'99^{150} = 0'22$.) Al realizar el aprendizaje de la red bayesiana, el modelo que más se ajusta a los datos dirá que $P(\neg s|e_c) = 1$ y $P(+s|e_c) = 0$, y según él, la probabilidad de E_c dada la presencia del síntoma es 0; es decir, el hallazgo $+s$ sería capaz de anular toda la evidencia a favor de E_c que pudieran aportar los demás hallazgos. Esto se debe a que ha habido un sobreajuste del modelo a los datos.

Sea un problema de tres variables, A, B y C, tal que B y C son condicionalmente independientes dado A, es decir, $P(b|a) \cdot P(c|a) = P(b, c|a)$. De ahí se deduce que las frecuencias observadas en la base de datos cumplirán aproximadamente la igualdad $N(+a, +b) \cdot N(+a, +c) = N(+a, +b, +c) \cdot N(+a)$. Sin embargo, es muy improbable que esta igualdad se cumpla exactamente; es decir, lo más probable es que aparezca una pequeña correlación accidental entre B y C (dado A) en la base de datos, a pesar de que en el mundo real son condicionalmente independientes. El modelo que mejor se ajustaría a dicha base de datos incluiría un enlace espurio $B \rightarrow C$ o $B \rightarrow C$, debido al sobreajuste.

Este último ejemplo nos muestra que las correlaciones espurias existentes en la base de datos pueden llevar a construir un modelo que contenga un enlace entre cada par de nodos, es decir, un grafo completo. Eso plantea tres problemas:

1. La falta de precisión, que ya hemos comentado en el primer ejemplo.

2. La pérdida de información: el modelo resultante no mostraría ninguna de las relaciones de independencia existentes en el mundo real. Por ejemplo, en los grafos obtenidos en el capítulo 4 se observan ciertas relaciones de independencia entre las variables, mientras que si trazáramos un enlace entre cada par de variables perderíamos esa información.

3. El tamaño del modelo: una red bayesiana de n variables basada en un grafo completo contendrá una tabla de probabilidad de n variables, cuyo tamaño será el mismo que el de la probabilidad conjunta, y además otra tabla de $n-1$ variables, otra de $n-2$, etc. En este caso, construir una red bayesiana es peor que representar la tabla de probabilidad conjunta explícitamente. Como el tamaño del problema crece de forma exponencial, con los ordenadores actuales sólo podríamos construir redes bayesianas de unas 25 o 30 variables. Sin embargo, luego veremos que hoy en día existen algoritmos de aprendizaje capaces de construir modelos con cientos de variables.

La forma habitual de evitar el sobreaprendizaje a la hora de construir el grafo de la red es tratar de construir modelos sencillos, es decir, con un pequeño número de enlaces. Más adelante veremos cómo lo consigue cada uno de los métodos de aprendizaje que vamos a estudiar.

2.2.2. Aprendizaje probabilista

Denominamos aprendizaje probabilista a aquél que se basa en los principios de la teoría de la probabilidad, independientemente de que el modelo aprendido sea de tipo probabilista (por ejemplo, una red bayesiana) o no probabilista (por ejemplo, una red neuronal). Dentro de él vamos a estudiar dos modalidades: el de máxima verosimilitud y el bayesiano.

Aprendizaje de máxima verosimilitud

La verosimilitud es una función, habitualmente representada por λ, que indica en qué medida cada hipótesis o cada modelo explica los datos observados:

$$\lambda(\text{modelo}) = P(\text{datos} \mid \text{modelo}) \tag{2.1}$$

El aprendizaje de máxima verosimilitud consiste en tomar la hipótesis o el modelo que maximiza esta función. Veámoslo con un ejemplo:

Deseamos construir un modelo que indique la probabilidad de supervivencia para cierta enfermedad. Se trata de un modelo muy simple, pues sólo tiene un parámetro, θ, la tasa de supervivencia. Sabemos que el verdadero valor de θ ha de estar entre entre 0 y 1. Para cada paciente individual,

$$\begin{cases} P(\text{supervivencia}) = \theta \\ P(\text{fallecimiento}) = 1-\theta \end{cases} \tag{2.2}$$

Supongamos ahora que tenemos una base de datos de n pacientes que sufren esa enfermedad, de los cuales han sobrevidido m, y a partir de ellos tratamos de estimar el valor del parámetro θ. (Como dicen Castillo et al. [7, pág. 505], "en el lenguaje de los estadísticos, el aprendizaje estadístico se llama *estimación*".) La verosimilitud para estos datos es[1]

$$\lambda(\theta) = P(\text{datos}|\theta) = \frac{n!}{m!(n-m)!}\,\theta^m\,(1-\theta)^{n-m} \tag{2.3}$$

La estimación de máxima verosimilitud para θ consiste en tomar el valor que maximiza $\lambda(\theta)$, que en este caso es $\theta = m/n$.[2]

Aprendizaje bayesiano

Uno de los tipos de aprendizaje desarrollados en el campo de la inteligencia artificial, inspirado en la estadística bayesiana, es el demominado *aprendizaje bayesiano*, que consiste en asignar una probabilidad a priori a cada uno de los modelos, $P(\text{modelo})$. La probabilidad a posteriori del modelo dados los datos se define mediante el teorema de Bayes:

$$P(\text{modelo} \mid \text{datos}) = \frac{P(\text{modelo}) \cdot P(\text{datos} \mid \text{modelo})}{P(\text{datos})} \tag{2.4}$$

Obsérvese la semejanza con los problemas de diagnóstico probabilista. En ellos se trata de diagnosticar la enfermedad que mejor explica los síntomas. Aquí tratamos de "diagnosticar" el modelo que mejor explica los datos observados. En ambos casos, el "diagnóstico" se basa en una probabilidad a priori, que representa el conocimiento que teníamos antes de observar las observaciones, y en una verosimilitud que indica en qué medida cada una de nuestras hipótesis (en este caso, cada modelo) explica los datos observados.

En la ecuación anterior el denominador es una constante, en el sentido de que es la misma para todos los modelos. Por tanto, si no queremos conocer la probabilidad absoluta, sino sólo cuál de los modelos tiene mayor probabilidad que los demás, podemos quedarnos con una versión simplificada de ella:

$$P(\text{modelo} \mid \text{datos}) \propto P(\text{modelo}) \cdot P(\text{datos} \mid \text{modelo}) \tag{2.5}$$

[1] La probabilidad $P(m|\theta)$ viene dada por la distribución de Bernouilli. El factor θ^m corresponde a la probabilidad de que sobrevivan m pacientes, $(1-\theta)^{n-m}$ es la probabilidad de que mueran los demás, y $n!/(m!(n-m)!)$ es el número de combinaciones de n elementos tomados de m en m.

[2] Para maximizar esta función, resulta más cómodo tomar su logaritmo neperiano (como el logaritmo es una función monótona, el máximo de λ es el mismo que el de $\ln\lambda$):

$$\ln\lambda(\theta) = \ln\frac{n!}{m!(n-m)!} + m\ln\theta + (n-m)\ln(1-\theta)$$

$$\frac{d}{d\theta}\ln\lambda(\theta) = m\frac{1}{\theta} - (n-m)\frac{1}{1-\theta}$$

$$\frac{d}{d\theta}\ln\lambda(\theta) = 0 \Longleftrightarrow \theta = \frac{m}{n}$$

Se puede comprobar además que la derivada segunda de $\ln\lambda(\theta)$ es negativa en todo el intervalo (0,1), y por tanto $\lambda(\theta)$ tiene un máximo en $\theta = m/n$.

El aprendizaje de máxima verosimilitud es un caso particular del aprendizaje bayesiano, pues cuando la probabilidad a priori es constante, entonces la probabilidad a posteriori es proporcional a la verosimilitud,

$$P(\text{modelo}) = \text{constante} \Longrightarrow P(\text{modelo} \mid \text{datos}) \propto P(\text{datos} \mid \text{modelo}) \tag{2.6}$$

y maximizar una de ellas es lo mismo que maximizar la otra. Veamos un par de ejmplos para aclararlo:

Supongamos que en el problema del ejemplo anterior la probabilidad a priori de θ es constante: $P(\theta) = c$. En este caso,

$$P(\theta|\text{datos}) \propto P(\theta) \cdot P(\text{datos}|\theta) \propto \theta^m \, (1-\theta)^{n-m} \tag{2.7}$$

El máximo de $P(\theta|m)$ es el mismo que el de $\lambda(\theta)$, es decir, $\theta = m/n$.

En cambio, si en lugar de tomar una probabilidad a priori constante tomamos $P(\theta) = c\ \theta^k(1-\theta)^l$, donde c es una constante de normalización. En este caso, la probabilidad a posteriori es:[3]

$$P(\theta|\text{datos}) \propto P(\theta) \cdot P(\text{datos}|\theta) \propto \theta^{k+m}(1-\theta)^{l+n-m} \tag{2.8}$$

El máximo de esta función corresponde al valor $\theta = (k+m)/(k+l+n)$.

En los dos ejemplos anteriores, la distribución $P(\theta)$ indica la probabilidad de un parámetro probabilista (una probabilidad), y por eso se dice a veces que $P(\theta)$ es una *probabilidad de segundo orden.*

Insistimos en que lo más característico del aprendizaje bayesiano es el hecho de asignar una probabilidad a priori a cada uno de los modelos. En los dos ejemplos anteriores, el modelo venía caracterizado por el parámetro θ, y por eso hemos identificado $P(\text{modelo})$ con $P(\theta)$. Recordamos también que el aprendizaje bayesiano **no** está especialmente relacionado con las redes bayesianas: como vamos a ver a continuación, hay métodos de aprendizaje de redes bayesianas que no tienen nada que ver con el aprendizaje bayesiano, y viceversa, puede aplicarse el aprendizaje bayesiano para construir modelos que no tienen nada que ver con las redes bayesianas; por ejemplo, una red neuronal.

2.3. Aprendizaje paramétrico

El aprendizaje paramétrico da por supuesto que conocemos la estructura (el grafo) de la red bayesiana y, en consecuencia, también conocemos la factorización de la probabilidad

[3] Quizá el lector se pregunte por qué hemos escogido esa probabilidad a priori. Observe que, de acuerdo con la ecuación (2.7), si la probabilidad a priori es constante y los datos observados nos dicen han sobrevivido k pacientes y han muerto l, la probabilidad a posteriori de θ es proporcional a $\theta^k(1-\theta)^l$. Por tanto, la probabilidad a priori considerada en este ejemplo podría proceder de tales datos.

Supongamos ahora que tenemos una segunda base de datos con m supervivientes y $(n-m)$ fallecidos. Para estos datos, la verosimilitud es proporcional a $\theta^m(1-\theta)^{n-m}$. La probabilidad a posteriori final es $\theta^{k+m}(1-\theta)^{l+n-m}$. Este resultado es coherente con el hecho de que en total ha habido $k+m$ supervivientes y $l+n-m$ fallecidos.

y cuáles son las probabilidades condicionadas que forman la red. En el caso de variables discretas, podemos considerar que cada probabilidad condicionada $P(x_i|pa(X_i))$ es un parámetro, que llamaremos $\theta_{P(x_i|pa(X_i))}$. Sin embargo, para cada configuración $pa(X_i)$ se cumple que

$$\sum_{x_i} P(x_i|pa(X_i)) = 1 \tag{2.9}$$

Por tanto, si X_i toma n_{X_i} valores, el número de parámetros independientes para cada configuración $pa(X_i)$ es $n_{X_i} - 1$.

Notación Dado que resulta engorroso en muchos casos escribir $\theta_{P(x_i|pa(X_i))}$, en la bibliografía sobre aprendizaje es habitual utilizar la notación θ_{ijk}, cuyo significado es el siguiente:

- i representa la variable X_i. Por tanto, si la red tiene n_I variables, se cumple que $1 \leq i \leq n_I$.
- j representa la j-ésima configuración de los padres de X_i. Si X_i tiene k padres binarios, entonces hay 2^k configuraciones de $Pa(X_i)$, lo cual implica que $1 \leq j \leq 2^k$.
- k representa el valor que toma la variable X_i. El k-ésimo valor de X_i se puede escribir como x_i^k. Si esta variable puede tomar n_{X_i} valores, entonces $1 \leq k \leq n_{X_i}$.

Con esta notación, la ecuación anterior se puede reescribir como

$$\forall i, \forall j, \quad \sum_{k=1}^{n_{X_i}} \theta_{ijk} = 1 \tag{2.10}$$

También es habitual que $\boldsymbol{\theta}$ denote el conjunto de parámetros de la red (todas las probabilidades condicionales), $\boldsymbol{\theta}_i$ el subconjunto de probabilidades condicionales asociadas a la familia de X_i, y $\boldsymbol{\theta}_{ij}$ el conjunto de probabilidades condicionadas correspondientes a la j-ésima configuración de $Pa(X_i)$.Observe que hemos escrito $\boldsymbol{\theta}$, $\boldsymbol{\theta}_i$ y $\boldsymbol{\theta}_{ij}$ en negrita porque representan conjuntos de parámetros, mientras que θ_{ijk} representa un único parámetro. Por tanto, en el caso de variables binarias, tenemos:

- Para cada configuración $Pa(X_i)$, $\boldsymbol{\theta}_{ij} = \{\theta_{ijk} \mid 1 \leq k \leq n_{X_i}\}$.
- Para cada variable X_i, $\boldsymbol{\theta}_i = \bigcup_{j=1}^{2^k} \boldsymbol{\theta}_{ij}$.
- Para la red, $\boldsymbol{\theta} = \bigcup_{i=1}^{n_I} \boldsymbol{\theta}_i$.

Cuando tenemos una base de datos, N_{ijk} representa el número de casos en que la variable X_i toma el valor x_i^k y los padres de X_i toman los valores correspondientes a la j-ésima configuración.

2.3.1. Aprendizaje (estimación) de máxima verosimilitud

Una forma sencilla de estimar cada uno de los parámetros es por el método de la máxima verosimilitud: si la base de datos contiene n casos en que las variables de $Pa(X_i)$ toman los valores correspondientes a la configuración $pa(X_i)$ y en m de esos casos la variable X_i toma el valor x_i, entonces la estimación del parámetro $\theta_{P(x_i|pa(X_i))}$ es $\hat{\theta}_{P(x_i|pa(X_i))} = m/n$. Con la notación introducida anteriormente, esto puede expresarse así:

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}} \tag{2.11}$$

Esta ecuación tiene dos problemas. El primero es que cuando $N_{ij} = 0$ (no hay ningún caso en la base de datos correspondiente a esa configuración de padres de X_i) entonces también $N_{ijk} = 0$ y por tanto el valor del cociente es una indeterminación. El otro problema es el sobreajuste, que ya ha sido comentado anteriormente. Estos dos problemas se resuelven mediante la estimación bayesiana, tal como vamos a ver en seguida.

2.3.2. Aprendizaje bayesiano

Una alternativa más compleja es utilizar el aprendizaje bayesiano, lo cual implica dar una distribución de probabilidad para los parámetros, $P(\boldsymbol{\Theta})$. Obsérvese que estamos utilizando la letra griega theta *en mayúscula* porque en el caso del aprendizaje bayesiano cada parámetro se trata como una variable aleatoria, es decir, una variable que tiene una distribución de probabilidad asociada.

La primera cuestión, por tanto, es determinar la forma de $P(\boldsymbol{\Theta})$ y la segunda obtener un algoritmo que permita estimar los valores de $\boldsymbol{\Theta}$ de forma eficiente, es decir, con un consumo de tiempo y de memoria razonables. Para poder abordar ambas cuestiones es necesario introducir hipótesis adicionales y restricciones sobre la forma de $P(\boldsymbol{\Theta})$. La mayor parte de los métodos propuestos en la literatura se basan en la hipótesis de *independencia de los parámetros*, que se expresa así:

$$P(\boldsymbol{\theta}) = \prod_i \prod_j P(\boldsymbol{\theta}_{ij}) \tag{2.12}$$

Hay autores que dividen esta hipótesis en dos partes: la primera es la *independencia global de los parámetros*, es decir que dos parámetros (las probabilidades condicionadas) de dos familias diferentes son independientes entre sí,

$$P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_i) \; ; \tag{2.13}$$

la segunda es la *independencia local de los parámetros*, es decir, que dentro de una familia los parámetros correspondientes a una configuración de los padres[4] son independientes de los correspondientes a las demás configuraciones:

$$P(\boldsymbol{\theta}_i) = \prod_j P(\boldsymbol{\theta}_{ij}) \tag{2.14}$$

[4]Si X_i es una variable binaria sólo se necesita un parámetro por cada configuración de los padres.

Luego hay que indicar qué forma tiene la distribución de probabilidad a priori para los parámetros de una configuración, $P(\boldsymbol{\theta}_{ij})$. Lo más frecuente en la literatura es suponer que se trata de una distribución de Dirichlet. En esta sección no vamos a entrar en los detalles del proceso (que pueden consultarse en el libro de Castillo et al. [7]), tan sólo vamos a comentar que cuando no hay conocimiento a priori es razonable asignar una probabilidad uniforme (constante) a todos los valores de los parámetros. En este caso, es decir, cuando tenemos una distribución de Dirichlet uniforme, el estimador de θ_{ijk} correspondiente al valor máximo de la probabilidad a posteriori es

$$\hat{\theta}_{ijk} = \frac{N_{ijk} + 1}{N_{ij} + n_{X_i}} \tag{2.15}$$

La diferencia de esta ecuación con la (2.11) es que sumamos 1 en el numerador, lo cual obliga a sumar n_{X_i} para que las probabilidades sumen la unidad (cf. ecuaciones (2.9) y (2.10)). Esta modificación se suele denominar *corrección de Laplace*. Obsérvese que el resultado es el mismo que si aplicáramos la estimación de máxima verosimilitud pero añadiendo a la base de datos un caso ficticio por cada configuración de la familia de X_i; es decir, para cada configuración de $Pa(X_i)$ se añaden n_{X_i} casos, uno por cada valor x_i^k de X_i.

Así se resuelven los dos problemas que presentaba el aprendizaje de máxima verosimilitud. El primero es que, aunque $N_{ij} = 0$, el cociente está definido, pues en ese caso $\hat{\theta}_{ijk} = 1/n_{X_i}$ para todo k. El segundo es que se evita el sobreajuste. Así en el ejemplo de la página 9, en vez de tener $\hat{P}(\neg s|e_c) = 1$ y $\hat{P}(+s|e_c) = 0$, tendríamos $\hat{P}(\neg s|e_c) = (1+1)/(1+2) = 2/3$ y $\hat{P}(+s|e_c) = (0+1)/(1+3) = 1/3$.

Existen otras variantes del método, similares a la corrección de Laplace, que en vez de añadir un caso ficticio por cada configuración de $Pa(X_i)$ y cada valor de X_i, añaden α_{ijk} casos, donde α_{ijk} puede ser un número no entero:

$$\hat{\theta}_{ijk} = \frac{N_{ijk} + \alpha_{ijk}}{N_{ij} + \alpha_{ij}} \tag{2.16}$$

donde α_{ij}, definido como $\alpha_{ij} = \sum_k \alpha_{ijk}$, se denomina *espacio muestral equivalente*, porque el resultado es el mismo que si hubiéramos realizado la estimación de los parámetros de $\boldsymbol{\theta}_{ij}$ mediante el método de máxima verosimilitud pero añadiendo α_{ij} casos. a la base de datos. Estos valores de α representan la información a priori, es decir, la probabilidad a priori de los parámetros.

En principio, los valores de las α's se podrían ajustar para reflejar las estimaciones subjetivas aportadas por los expertos. Sin embargo, en la práctica nunca (o casi nunca) hay expertos capaces de aportar información a priori, por lo que lo más habitual es aplicar la corrección de Laplace (es decir, $\alpha_{ijk} = 1$ en todos los casos) o bien tomar un valor de corrección menor, por ejemplo $\alpha_{ijk} = 0'5$, que es como una "corrección de Laplace suavizada". En la literatura sobre el tema a veces se muestran los resultados experimentales obtenidos con diferentes valores de esta corrección.

2.4. Aprendizaje estructural a partir de relaciones de independencia

Los primeros algoritmos de aprendizaje estructural de redes bayesianas que surgieron estaban basados en un análisis de las relaciones de dependencia e independencia presentes en la distribución de probabilidad P: el problema consiste en encontrar un grafo dirigido acíclico (GDA) que sea un mapa de independencias (I-mapa) de P. En realidad, buscamos un I-mapa minimal; es decir, si hay dos grafos que sólo se diferencian en que hay un enlace que aparece en el primero pero no en el segundo y ambos son I-mapas de P preferiremos el segundo, por cuatro motivos: porque muestra más relaciones de independencia que el primero, porque va a necesitar menos espacio de almacenamiento (alguna de las tablas será más pequeña), porque va a ser más preciso (al necesitar menos parámetros podrá estimarlos con mayor fiabilidad, reduciendo además el riesgo de sobreajuste) y porque conducirá a una computación más eficiente. Una vez obtenido el grafo, ya se puede realizar el aprendizaje paramétrico para hallar las probabilidades condicionadas y completar así la red bayesiana.

Obtención de las relaciones de dependencia e independencia En la práctica nunca conocemos P (la distribución de probabilidad del mundo real), sino un conjunto de casos obtenidos del mundo real y recogidos en una base de datos. Por eso, el primer problema que debe afrontar este método es cómo obtener las relaciones de dependencia e independencia de P a partir de la base de datos. A primera vista podríamos pensar que cuando dos variables están correlacionadas en la base de datos es porque están correlacionadas en P. Sin embargo, también es posible que se trate de una correlación espuria, es decir, accidental. Para ello se suelen aplicar los test de independencia de la estadística clásica, es decir, se realiza un *contraste de hipótesis* para discernir estas dos posibilidades:

- **Hipótesis experimental**, H_E: Las variables están correlacionadas.
- **Hipótesis nula**, H_0: Las variables son independientes, es decir, la correlación que se observa en la base de datos se debe al azar.

Luego se aplica un test χ^2 para determinar la probabilidad de que siendo cierta la hipótesis nula, H_0, se produzca por azar una correlación tan grande como la observada entre las variables (u otra mayor). Esta probabilidad se denomina p. Si p es inferior a cierto umbral α, conocido como *nivel de significancia*, se rechaza la hipótesis nula, es decir, se concluye que realmente las variables están correlacionadas. En el problema que nos ocupa, eso lleva a incluir una relación de dependencia como dato de entrada para el aprendizaje estructural. En cambio, si $p \geq \alpha$, se incluye una relación de independencia.

La dificultad de esta búsqueda de dependencias e independencias es que el número de relaciones posibles crece super-exponencialmente con el número de variables, pues hay que examinar cada relación del tipo $I_P(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$, con la única condición de que los tres subconjuntos sean disjuntos y $\mathbf{X}$ e $\mathbf{Y}$ sean no vacíos. Para cada relación hay que realizar tantos tests de independencia como configuraciones existen para $\mathbf{X}$, $\mathbf{Y}$ y $\mathbf{Z}$; al menos, hay que hacer todos los test necesarios hasta encontrar una combinación de configuraciones

en que el test determine que hay correlación entre **X** e **Y** dado **Z**, lo cual nos llevaría a incluir $\neg I_P(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ en la lista de relaciones.

Por este motivo, el aprendizaje basado en relaciones sólo puede utilizarse para problemas en que el número de variables es muy reducido.

Construcción del grafo Una vez obtenida la lista de relaciones, hay que construir el grafo. El algoritmo más conocido es el denominado PC, de Spirtes et al. [33, 34]. Como puede verse en el algoritmo 1, el algoritmo consta de dos fases. En la primera, toma como punto de partida un grafo completo no dirigido y va eliminando enlaces basándose en las relaciones de independencia. La segunda fase consiste en asignar una orientación a los enlaces del grafo no dirigido obtenido en la primera fase. Los detalles de este algoritmo, así como su justificación teórica, pueden encontrarse en las referencias citadas y en el libro de Neapolitan [25, cap. 10], que estudia además otros algoritmos de aprendizaje estructural similares.

Algoritmo 1: Algoritmo PC

Input: conjunto de nodos V y conjunto de relaciones de independencia IND
Result: estructura de la red aprendida

```
1  //Primera etapa:
2  Crear el grafo no dirigido completo gp
3  i = 0
4  while |ADJ_X| > i ∀X ∈ V do
5      foreach X ∈ V do
6          foreach Y ∈ ADJ_X do
7              Encontrar un subconjuntoconjunto S ⊆ ADJ_X − Y tal que |S| = i y
               I(X, Y|S) ∈ IND if existe ese subconjunto then
8                  S_XY = X eliminar la arista X - Y de gp
9      i = i + 1
10 //Segunda etapa:
11 foreach enlace del tipo X - Z - Y do
12     if Z ∉ S_XY then
13         orientar X - Z - Y como X → Z ← Y
14 while queden enlaces sin orientar do
15     foreach enlace del tipo X → Z − Y do
16         orientar Z - Y como Z ← Y
17     foreach enlace del tipo X - Y tal que hay un camino de X a Y do
18         orientar X - Y como X ← Y
19     foreach enlace del tipo X - Z - Y tal que existe W con X → W, Y → W, y Z -
       W do
20         orientar Z - W como Z ← W
```

2.5. Aprendizaje estructural mediante búsqueda heurística

Un método alternativo de aprendizaje estructural consiste en utilizar una métrica para determinar cuál es el mejor modelo. La primera dificultad que encontramos es que el número de modelos es infinito. Por ello descomponemos el problema en dos partes: buscar el mejor grafo y buscar los mejores parámetros para cada grafo posible. La segunda parte es el llamado aprendizaje paramétrico, que como ya hemos visto, puede resolverse en un tiempo razonable introduciendo algunas hipótesis. Por tanto, el problema se reduce a examinar todos los grafos posibles —su número es finito— y realizar un aprendizaje paramétrico para cada uno de ellos. Sin embargo, esta propuesta sólo es válida para problemas con muy pocas variables, porque el número de grafos posibles crece de forma super-exponencial con el número de variables. La solución es aplicar el concepto de *búsqueda heurística* desarrollado en el campo de la inteligencia artificial: dado que no es posible examinar todas las posibles soluciones, sino sólo una parte muy pequeña de ellas, vamos a realizar un proceso de búsqueda que consiste en generar unas pocas posibles soluciones, seleccionar la mejor (o las mejores), y a partir de ella(s) generar otras nuevas, hasta encontrar una que satisfaga ciertos criterios. Este proceso de *búsqueda en calidad*[5] necesita ser guiado por una *métrica* (en inglés, *score*) que indique la calidad de cada posible solución.

Por tanto, cada algoritmo de aprendizaje de este tipo se caracteriza por dos elementos:

- Una medida de calidad, que se usa para decidir cuál es la mejor de un conjunto de redes bayesianas. Esto es una medida global de calidad, ya que mide tanto la calidad de la estructura gráfica como la de los parámetros estimados.
- Un algoritmo de búsqueda, que se usa para recorrer el espacio de búsqueda atendiendo a unos criterios de calidad. como hemos indicado, el número de posibles redes, incluso para pocas variables, es extremadamente grande y es necesario tener un algoritmo que guíe la búsqueda a través de ese espacio ingente.

En los dos próximos apartados se tratan en profundidad cada uno de estos dos elementos y sus distintos subtipos.

2.5.1. Métricas de calidad

Una medida de calidad es un criterio mediante el cual se puede ordenar un conjunto de redes bayesianas según su calidad. Algunas propiedades deseables que deben satisfacer son:

- Dos redes que conducen a la misma estructura de independencia han de obtener el mismo valor de calidad.
- A las redes recomendadas por los expertos se les debe asignar calidades más altas que a las rechazadas por ellos.

[5] En el campo de la inteligencia artificial se distinguen tres tipos principales de búsqueda: búsqueda en profunidad (en inglés, *depth-first search*), búsqueda en anchura (*breadth-first search*) y búsqueda en calidad (*best-first search*).

- Las representaciones perfectas deben recibir calidades mayores que las imperfectas.
- Las I-representaciones mínimas deben recibir calidades mayores que las no mínimas.
- Las redes con reducido número de parámetros a igualdad del resto de propiedades deben recibir calidades mayores que las de elevado número de parámetros.
- A las redes que confirmen la información contenida en los datos debe asignársele una calidad mayor que a aquellas que contradigan éstos.

En la literatura existente se han propuesto varias medidas de calidad para redes bayesianas, que pueden clasificarse en tres tipos:

1. Métricas bayesianas.
2. Métricas de longitud mínima de descripción.
3. Métricas de información.

Cada uno de estos tipos se discuten en las secciones siguientes.

Métricas bayesianas

En estas métricas la calidad de una red se identifica con su probabilidad a posteriori, dada por la ecuación (2.5). La probabilidad a priori de una red es el resultado de la probabilidad de su grafo y de la probabilidad de los parámetros dado el grafo:

$$P(\text{red}) = P(\text{grafo}) \cdot P(\text{parámetros} \mid \text{grafo}) \tag{2.17}$$

Las métricas bayesianas se diferencian unas de otras en la forma de asignar estas dos distribuciones de probabilidad, $P(\text{grafo})$ y $P(\text{parámetros} \mid \text{grafo})$. Esta última es la probabilidad a priori de los parámetros dado el grafo. No vamos a hablar ahora de esta cuestión porque ya la hemos discutido en la sec. 2.3 (aprendizaje paramétrico bayesiano).

En cuanto a $P(\text{grafo})$, hay varias posibilidades. Por ejemplo, la métrica K2 [9], que es una de las más conocidas, supone que todos los grafos tienen la misma probabilidad a priori. Otra posibilidad sería dar mayor probabilidad a las redes que tienen menos enlaces y menos padres en cada familia, con el fin de evitar el sobreajuste.[6]

Por último, hay que especificar la probabilidad $P(\text{datos} \mid \text{red})$, para lo cual también es necesario introducir hipótesis adicionales. En particular, es habitual suponer que la base de datos está completa —es decir, no hay datos ausentes— y que los casos de la base de datos son condicionalmente independientes dado el modelo.

[6] En el algoritmo K2, que utiliza la métrica K2, el sobreajuste se evita de otra forma: limitando el número de padres que puede tener cada nodo.

En resumen, la métrica resultante, que, como hemos dicho, identifica la calidad de la red con su probabilidad a posteriori, viene dada por

$$\begin{aligned} P(\text{red} \mid \text{datos}) &\propto P(\text{red}) \cdot P(\text{datos} \mid \text{red}) \\ &= P(\text{grafo}) \cdot P(\text{parámetros} \mid \text{grafo}) \cdot \\ &\cdot P(\text{datos} \mid \text{grafo, parámetros}) \end{aligned}$$

En concreto, la métrica K2, propuesta por Cooper y Herskovits en 1992 [9](de ahí que esta métrica sea también conocida como métrica de Cooper-Herskovits) tiene la forma:

$$\begin{aligned} Q_{K2}(D,S) &= log(p(D)) + \\ &+ \sum_{i=1}^{n} \left[\sum_{j=1}^{s_i} \left[log \frac{\Gamma(r_i)}{\Gamma(N_{ij}+r_i)} + \sum_{k=0}^{r_i} log\Gamma(N_{ijk}+1) \right] \right] \end{aligned} \tag{2.18}$$

donde $\Gamma(.)$ es la función *gamma*, s_i es el número de posibles configuraciones de los padres y r_i es el número de valores distintos de X_i. Nótese que esta métrica no incluye ningún término explícito de penalización para la complejidad de la red, lo cual puede llevar a redes demasiado complejas (con el problema de sobreajuste que ello conlleva) a menos que el algoritmo de búsqueda usado se preocupe de limitar de alguna manera dicha complejidad.

Otra de las métricas bayesianas más utilizadas es la propuesta por Geiger y Heckerman en 1995 [18]:

$$\begin{aligned} Q_{GH}(D,S) &= log(p(D)) + log \int p(S|D,\theta) p(\theta|D) d\theta \\ &= log(p(D)) + \\ &+ \sum_{i=1}^{n} \left[\sum_{j=1}^{s_i} \left[log \frac{\Gamma(\eta_{ij})}{\Gamma(N_{ij}+\eta_{ij})} + \sum_{k=0}^{r_i} log \frac{\Gamma(N_{ijk}+\eta_{ijk})}{\Gamma(\eta_{ijk})} \right] \right] \end{aligned} \tag{2.19}$$

donde η_{ijk} son los parámetros de la distribución de Dirichlet de la probabilidad a priori de cada una de las configuraciones[7]. Como puede verse, esta medida tampoco incluye un término de penalización explícita de la complejidad de la red.

Como alternativa a 2.18 y 2.19, surge la medida de calidad bayesiana usual, que se obtiene evaluando la verosimilitud en el punto modal "a posteriori", y considerando un término de penalización explícito. La moda posterior es

$$\hat{\theta}_{ijk} = \frac{\eta_{ijk} + N_{ijk} - 1}{\eta_{ij} + N_{ij} - r_i} \tag{2.20}$$

Asintóticamente, 2.20 tiende a

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}} \tag{2.21}$$

[7]Como comentamos en la sección 2.3.2, usualmente se supone una distribución de Dirichlet para la probabilidad de los parámetros de cada configuración. Pueden verse más detalles en el libro de Castillo et al. [7].

Por ello, sustituyendo 2.20 en la verosimilitud, se obtien la medida Bayesiana estándar o usual:

$$\begin{aligned} Q_{SB}(D,S) = & log(p(D)) + \\ & + \sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i}(N_{ijk}+\eta_{ijk}-1)log\left(\frac{\eta_{ijk}+N_{ijk}-1}{\eta_{ij}+N_{ij}-r_i}\right) \\ & - \frac{1}{2}Dim(B)logN \end{aligned} \tag{2.22}$$

donde Dim(B) es la dimensión (el número de parámetros necesarios para especificar completamente la función de probabilidad conjunta de X) de la red bayesiana B y N es el tamaño de la muestra. El último término en 2.22 es un término que penaliza las redes con mayor número de parámetros.

Alternativamente, se puede utilizar la estimación asintótica 2.21 en vez de la 2.20 y obtener:

$$\begin{aligned} Q_{SB}(D,S) = & log(p(D)) + \sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i}N_{ijk}log\frac{N_{ijk}}{N_{ij}} \\ & - \frac{1}{2}Dim(B)logN \end{aligned} \tag{2.23}$$

Métricas de longitud mínima de descripción

Las medidas de calidad descritas hasta ahora requieren especificar la información "a priori", la estructura gráfica y la probabilística, pero esta información puede no ser accesible. Las medidas de longitud mínima de descripción son una forma alternativa de medir la calidad de una estructura de red. Estas métricas se basan en la posibilidad de representar el modelo y los datos en forma codificada. La codificación del modelo ocupa más espacio cuanto más complejo sea el modelo y la representación de los datos será más breve cuanto más cerca esté el modelo de los datos. El concepto de mínima longitud de descripción (LMD; en inglés, *minimum description length*, MDL) se refiere a la descripción más breve posible, es decir, utilizando la mejor codificación.

Al identificar la calidad de un modelo con su LMD cambiada de signo se atienden dos objetivos: por un lado, se asigna mayor calidad a los modelos más simples, lo cual es un antídoto contra el sobreajuste; por otro, se valoran más los modelos que más se ajustan a los datos. Buscar las redes bayesianas de mayor calidad es lo mismo que buscar las de menor LMD. En el caso de las redes bayesianas, la longitud de descripción incluye:

1. La longitud requerida para almacenar la estructura de la red bayesiana. Puesto que el número máximo de aristas en una red bayesiana con n nodos es $n(n-1)/2$, y se puede almacenar un 1 si existe la arista y un 0 en otro caso[8], entonces el máximo número de aristas (longitud) requerido es $n(n-1)/2$. Nótese que este número no

[8]Esta puede no ser la forma óptima de almacenar la estructura. Se ha elegido por razones de simplicidad.

depende de la estructura en particular. Por ello, no necesita ser incluída en la medida de calidad.

2. La longitud requerida para almacenar los parámetros θ. Es bien sabido que la longitud media necesaria para almacenar un número en el rango 0 a N es $\frac{1}{2}logN$. Por ello, para almacenar los parámetros de la función de probabilidad conjunta, se necesita una longitud de $\frac{1}{2}Dim(B)logN$, donde N es el tamaño de la muestra y

$$Dim(B) = \sum_{i=0}^{n}(r_i - 1) \prod_{X_j \in \Pi_i} r_j = \sum_{i=1}^{n}(r_i - 1)s_i \tag{2.24}$$

es el número de parámetros libres (grados de libertad) asociados a la función de probabilidad conjunta. Nótese que s_i es el número de parámetros libres (grados de libertad) asociados a la distribución de probabilidad condicional para X_i, $p(x_i|\pi_i)$, es decir, el número de posibles configuraciones de los padres de X_i.

3. La longitud de descripción del conjunto de datos S comprimidos usando la distribución asociada a (D, P), que en el caso de las redes bayesianas resulta ser

$$\sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i} N_{ijk} log \frac{N_{ijk}}{N_{ij}} \tag{2.25}$$

que es $-N$ veces la entropía

$$H(S, B) = \sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i} -\frac{N_{ijk}}{N} log \frac{N_{ijk}}{N_{ij}} \tag{2.26}$$

Bouckaert (1995) añade un término adicional para incorporar la información "a priori"' y propone la siguiente medida de mínima longitud de descripción

$$\begin{aligned} Q_{MLD}(B, S) =& log(p(D)) + \sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i} N_{ijk} log \frac{N_{ijk}}{N_{ij}} \\ & - \frac{1}{2} Dim(B) logN \end{aligned} \tag{2.27}$$

Nótese que las medidas de calidad 2.23 y 2.27 son las mismas, y que las medidas de calidad 2.22 y 2.27 son asintóticamente equivalentes.

Medidas de información

Otra forma de medir el ajuste entre la red y los datos consiste en calcular la *información mutua*, cuyo valor numérico se determina a partir de la teoría de la información. El problema de identificar la calidad de la red con la información mutua es que los modelos más complejos permiten alcanzar valores más altos en esta métrica, lo cual lleva al sobreajuste. Para evitarlo, se añade a la métrica un término adicional, denominado *penalización*, que resta un valor más alto cuanto mayor es la complejidad del modelo. Esto

conduce a la medida de información

$$Q_I(B,S) = log(p(D)) + \sum_{i=1}^{n}\sum_{j=1}^{s_i}\sum_{k=0}^{r_i} N_{ijk} log\frac{N_{ijk}}{N_{ij}} - Dim(B)f(N) \qquad (2.28)$$

donde f(N) es una función de penalización no negativa.

En la literatura existente se han propuesto muchas funciones de penalización, tales como *el criterio de máxima verosimilitud de información* ($f(N) = 0$), el *criterio de información de Akaike* [1], ($f(N) = 1$, esta métrica también es conocida como la métrica AIC) y el *criterio de información de Schwarz* [31], ($f(N) = log(N)/2$). Nótese que la medida de mínima longitud de descripción 2.27 es un caso particular de esta medida.

2.5.2. Algoritmos de búsqueda

Como hemos dicho ya, en la práctica es imposible examinar todos los grafos posibles y realizar un aprendizaje paramétrico para cada uno de ellos. Por eso se utilizan técnicas de búsqueda heurística. Se hace notar que hay métodos de búsqueda que trabajan en el espacio de todas las redes bayesianas y otros que lo hacen en el de las clases de equivalencia de las estructuras de red. Para ampliar detalles se remite al lector a los trabajos de Spirtes y Meek [35] y Chickering [8].

El primer algoritmo de búsqueda propuesto en la literatura fue K2 [9]. El punto de partida es un grafo vacío. En cada iteración, el algoritmo considera todos los enlaces posibles, es decir, todos aquellos que no forman un ciclo, y añade aquél que conduce a la red bayesiana de mayor calidad.[9] El algoritmo termina cuando no se pueden añadir más enlaces (el número de padres por nodo está acotado, con el fin de evitar el sobreajuste) o cuando no es posible aumentar la calidad de la red añadiendo un enlace. El algoritmo 2 muestra su funcionamiento.

Uno de los problemas del algoritmo K2 es que requiere una ordenación previa de los nodos. Para evitar este problema surge el *algoritmo B*, propuesto por Buntine en 1991 [5]. La única diferencia con el algoritmo K2 es que se usa una matriz en la que se almacenan los distintos incrementos provocados en la medida de calidad al añadir un enlace (en la casilla $A[i,j]$ se almacena el incremento obtenido al añadir el enlace del nodo i-ésimo al j-ésimo). El algoritmo 3 muestra su funcionamiento.

Otro de los algoritmos más utilizados es el *algoritmo del gradiente*, también conocido como algoritmo "Hill Climber". Su funcionamiento se muestra en el algoritmo 4. Partiendo de un grafo vacío (o de cualquier grafo acíclico), en cada iteración del algoritmo se realiza la operación (añadir o eliminar un enlace) que maximiza la puntuación de la red, siempre que se mejore la de la red obtenida en el paso anterior. El algoritmo para cuando ninguna de las operaciones posibles mejora la puntuación de la red anterior. Véase que en cada paso si

[9] En el artículo original de Cooper and Herskovits [9], la métrica utilizada era K2. Por eso la métrica y el algoritmo de búsqueda llevan el mismo nombre. Naturalmente, es posible utilizar el algoritmo de búsqueda K2 con cualquier otra métrica y, recíprocamente, utilizar la métrica K2 con cualquier otro algoritmo de búsqueda.

Algoritmo 2: Algoritmo K2

Input: variables que participan en el aprendizaje, conjunto de casos
Result: estructura de la red aprendida

```
1  //Etapa de iniciación:
2  Ordenar las variables
3  foreach variable do
4      Π_i ← φ
5  //Etapa iterativa:
6  for i=1 a n do
7      while δ > 0 y |Π_i| < número máximo de padres do
8          seleccionar el nodo Y ∈ X_1, ..., X_{i-1}
9          Π_i que maximiza la métrica
10         δ ← métrica(Π_i ∪ Y) - métrica(Π_i)
11         if δ > 0 then
12             Π_i ← Π_i ∪ Y
```

Algoritmo 3: Algoritmo B

Input: variables que participan en el aprendizaje, conjunto de casos
Result: estructura de la red aprendida

```
1  //Etapa de iniciación:
2  foreach variable do
3      Π_i ← φ
4  for i=1 a n do
5      for j=1 a n do
6          if i ≠ j then
7              A[i, j] ← m_i(X_j) − m_i(φ)
8          else
9              A[i, j] ← −∞ //no permitimos X_i → X_i
10 //Etapa iterativa:
11 for i=1 a n do
12     while A[i, j] > 0yA[i, j] ≠ −∞, ∀i, j do
13         if A[i, j] ¿0 then
14             Π_i → Π_i ∪ X_j  for X_a ∈ Ascen_i, X_b ∈ Descen_i do
15                 A[a, b] ← −∞ //no permitimos ciclos
16             for k = 1 a n do
17                 if A[i, k] > −∞ then
18                     A[i, k] ← m_i(Π_i ∪ X_k) − m_i(Π_i)
```

una arista con destino en el nodo X_i es añadida o eliminada, sólo es necesario reevaluar la puntuación de ese nodo en concreto. Los algoritmos de este tipo, que necesitan recalcular localmente algunas puntuaciones para obtener la puntuación de la siguiente red, se dice que tienen *actualización local de la puntuación* y, obviamente, son considerablemente más eficientes que aquellos que no la tienen.

Algoritmo 4: Algoritmo del gradiente

```
   Input: Red inicial, árbol de casos y métrica
   Result: red aprendida
1  while ha habido mejora do
2      Se obtiene la lista de operaciones que se pueden realizar.
3      foreach operacion do
4          La métrica calcula la puntuación de la red resultante de aplicar la operación.
5          if es mejor que la mejor puntuación then
6              Almacenamos la puntuación y la edición.
7      if ha habido mejora then
8          Realiza la mejor de las ediciones.
```

El principal problema de estos métodos es que se trata de *algoritmos voraces*, por lo que una vez añadido un enlace nunca se borra. Eso hace que la probabilidad de quedar atrapado en un máximo local sea muy alta.

Una forma de reducir (no de eliminar) este problema consiste en dar mayor flexibililidad al algoritmo, permitiendo que en cada paso se realice una de estas operaciones:

1. Añadir un enlace (siempre que no cree un ciclo).
2. Borrar un enlace.
3. Invertir un enlace (siempre que no se cree un ciclo).

El precio que se paga para reducir la probabilidad de máximos locales es una mayor complejidad computacional, pues el número de grafos que hay que examinar es mucho mayor. Otras posibles soluciones para el problema de los óptimos locales pasan por usar una mejora al algoritmo del gradiente conocida como *algoritmo del gradiente iterado* que consiste en introducir una pequeña modificación en el grafo obtenido por el algoritmo del gradiente y volver a aplicar el algoritmo partiendo de ese nuevo grafo repetidas veces. Además, existen otros métodos alternativos como *"simulated annealing"* ([24]) o la búsqueda en calidad propuesta por Korf en 1993 [22].

Concluimos esta sección señalando que existen métodos híbridos de aprendizaje que combinan la búsqueda heurística con la detección de relaciones de dependencia e independencia.

2.6. Otras cuestiones

Para terminar, vamos a mencionar cuatro cuestiones muy importantes que no han sido abordadas en este trabajo pero pueden ser estudiadas en profundidad consultando las referencias aportadas.

1. **Datos incompletos.** Los métodos que hemos descrito en este capítulo suponen que la base de datos es completa, es decir, que no tiene valores ausentes. Sin embargo, en la práctica la mayor parte de las bases de datos no cumplen esta condición. Algunos de los métodos más utilizados para abordar este problema son:

   - El muestreo de Gibbs (véanse, por ejemplo, [19, 6, 20] para una descripción detallada del mismo[10]). Se basa en simular secuencialmente las distribuciones univariadas de las variables dadas todas las demás.
   - El algoritmo EM para datos incompletos. Consta de dos etapas: la primera es la etapa de cálculo de los valores esperados, en la que se calcula la esperanza de los datos incompletos o funciones de ellos. En la segunda etapa se maximiza cierta función. Las dos etapas se iteran hasta conseguir la convergencia del proceso, que está garantizada bajo ciertas condiciones de regularidad [10].

2. **Aprendizaje de redes causales.** Como sabemos, algunos grafos admiten una interpretación causal y probabilista, mientras que otros sólo admiten la interpretación probabilista. La mayor parte de los métodos de aprendizaje sólo garantizan que las redes bayesianas aprendidas puedan ser interpretadas en sentido probabilista. Como introducción a los métodos de aprendizaje que permiten obtener conclusiones causales, recomendamos el libro de Neapolitan [25, secs. 6.5, 7.1.5 y 8.3], en especial las secciones 1.5, 2.6 y 11.4.2 y todo el capítulo 10. Un tratamiento más extenso se encuentra en los libros de Spirtes et al. [34], Glymour y Cooper [21] y Pearl [29].

3. **Variables ocultas.** Habitualmente, la red bayesiana contiene las mismas variables que la base de datos a partir de la cual ha sido construida. Sin embargo, en algunos casos la distribución de probabilidad asociada a la base de datos puede contener algunas relaciones de dependencia e independencia que no pueden ser modeladas adecuadamente mediante un GDA. Eso puede hacernos sospechar la presencia de una variable oculta (en inglés, *hidden variable* o *latent variable*) que induce tales relaciones. Algunos métodos para construir redes bayesianas que incluyen variables ocultas pueden verse en [25, sec. 8.5].

4. **Clasificadores basados en redes bayesianas.** Los métodos de aprendizaje descritos en este capítulo tratan de construir una red bayesiana cuya probabilidad conjunta se asemeje lo más posible a la distribución de probabilidad del mundo real. Por eso las métricas de calidad descritas en las secciones anteriores miden, sobre todo, el ajuste entre la red bayesiana y los datos. Sin embargo, en la mayor parte de las aplicaciones prácticas las redes bayesianas se utilizan como clasificadores. Por ejemplo, para determinar qué enfermedad padece una persona, para detectar qué avería tiene una máquina, para predecir si un cliente va a devolver el préstamo solicitado, para

[10] Para una descripción sucinta véanse: [7, sec. 11.10] y [25, secs. 6.5, 7.1.5 y 8.3].

distinguir el correo interesante del correo basura, etc. Ahora bien, la red que mejor clasifica no es necesariamente la que mejor modeliza la probabilidad. Por ello, encontrar el mejor clasificador basado en una red bayesiana es un problema diferente del aprendizaje en general. El lector interesado en el tema puede encontrar abundantes referencias en Internet introduciendo el término "Bayesian network classifiers".

Capítulo 3

Diseño e implementación

Los ordenadores siempre se empeñan en hacer lo que les decimos que hagan en lugar de hacer lo que queremos que hagan.

Anónimo

En este capítulo se hace un recorrido por el diseño y la implementación del módulo de aprendizaje centrándose en sus tres elementos clave: los algoritmos de aprendizaje, las métricas y el almacenamiento de bases de datos. En las secciones 3.1 y 3.3 se explica el diseño general de un algoritmo de aprendizaje y de una métrica, respectivamente, y en la sección 3.2 se muestra un ejemplo de implementación de un algoritmo, en este caso el *algoritmo del gradiente*. A continuación, en la sección 3.4 se habla del almacenamiento en memoria de los casos de una base de datos, explicando los algoritmos de almacenamiento y de cálculo de frecuencias absolutas de las distintas configuraciones de variables. Por último, en la sección 3.5 se comentan algunos aspectos relativos al diseño e implementación de la interfaz gráfica. Tanto la estructura general de las dos primeras secciones como las explicaciones de los patrones *Observer* (pág. 35) y *Command* (pág. 38), y sus modificaciones han sido tomadas de la Tesis (aún no publicada) del Profesor Manuel Arias.

3.1. Algoritmos de aprendizaje en Carmen

Un algoritmo es una lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema. En programación, lo normal es implementar un algoritmo como un método de una clase, que recibe datos de entrada, sigue un proceso y proporciona datos de salida. Sin embargo, en nuestro caso, este esquema sencillo se ha alterarado para introducir algunos requisitos de control impuestos por Carmen con el objetivo de cumplir las especificaciones marcadas en el diseño de esta herramienta. El resultado ha sido un patrón de diseño de algoritmos que se ha utilizado para programar algoritmos especificos de aprendizaje de redes bayesianas.

El módulo de aprendizaje desarrollado se centra en los algoritmos de aprendizaje me-

diante búsqueda heurística (un análisis teórico de los mismos puede verse en la sección 2.5) por ser los más utilizados en la actualidad (recuérdese que el aprendizaje a partir de relaciones de independencia está muy limitado por el crecimiento super-exponencial de dichas relaciones). Aunque en la versión actual se incluye un único algoritmo (el *algoritmo del gradiente*), el desarrollo de otras soluciones es muy sencillo debido a que el diseño de la herramienta Carmen en general y de este módulo de aprendizaje en particular se ha realizado con el objetivo prioritario de facilitar su ampliación.

3.1.1. Análisis de requisitos

Dado que el módulo de aprendizaje se enmarca dentro de la herramienta Carmen, existen una serie de requisitos generales que son necesarios para conseguir el objetivo global de implantación entre la comunidad científica. Algunos de estos requisitos, que ya han sido comentados en la introducción, son la robustez y fiabilidad necesarias para formar parte de nuevas aplicaciones, la escalabilidad de los algoritmos o la facilidad de ampliación.

El requisito más importante es que una tercera persona que haga alguna aportación a Carmen no debe encontrarse con dificultades añadidas debido a las consideraciones que se verán en el modelo de diseño, es decir, los algoritmos deben centrarse únicamente en los aspectos relativos a los modelos probabilistas sobre los que operan.

El hecho de que se quiera preservar la capacidad de ampliación del módulo de aprendizaje por terceras personas, hace necesario prestar especial atención a la documentación generada en las distintas etapas del proyecto. Tanto la documentación presente en el código fuente como los manuales que acompañan a dicho código han de ser generados con el objetivo principal de facilitar la ampliación de los distintos componentes.

Por otra parte, a diferencia de los algoritmos de inferencia, en este caso no existe interacción con el usuario, de modo que se evitan los problemas derivados de la interactividad. Sin embargo, es necesario integrar varios procesos que van a operar sobre una misma estructura de datos,[1] lo cual va a suponer un problema de coordinación y de garantización de la consistencia de dichos datos.

Un último requisito es que se desea dejar constancia de las operaciones que se van realizando en un archivo de anotaciones. Se desea que el nivel de detalle de la descripción sea configurable para permitir explicaciones más o menos profundas de las acciones realizadas.

3.1.2. Modelo de análisis

El modelo de análisis trata de representar la estructura global del sistema, describe la realización de casos de uso, sirve como una abstracción del modelo de diseño y se centra en los requerimientos no funcionales. Este modelo de análisis no es un diagrama final que describe todos los posibles conceptos y sus relaciones, sino un primer intento por definir los conceptos claves que describen el sistema.

[1] En principio de un modo secuencial, pero no se descarta la concurrencia.

La primera decisión que se tuvo que tomar fue cómo representar un algoritmo. Dado que las necesidades expuestas en el apartado anterior son bastante amplias se ha llegado a la conclusión de que un algoritmo tiene sufiente entidad como para ser modelado como una clase (en este caso, la clase *LearningAlgorithm*).

La representación del modelo gráfico probabilista se deja en manos de la clase *Network*, que en el modelo de diseño se dividirá en varios tipos de clases con otras muchas asociadas (entre ellas, la clase *ProbNet*, que es la que maneja directamente el algoritmo de aprendizaje) pero en éste nivel más abstracto sólo interesa *qué* se hace y *quién* lo hace. La representación en pantalla de la red aprendida corre a cargo de la clase *GUI*.

Para dejar un registro de las actividades hemos creado la clase *LogFile*, que simplemente se encarga de escribir en un fichero de texto (Carmen.log) los pasos que se vayan ejecutando.

Otro de los módulos importantes es el encargado del almacenamiento y manejo de los datos cargados a partir de un fichero de casos. Dado que el manejo de estos casos puede ser complejo y requiere de una serie de algoritmos especificos[2] se ha tomado la decisión de implementar una clase que modele una estructura de árbol capaz de gestionar los datos de forma eficiente, la clase *CaseTree*.

Por último, la clase encargada de iniciar todo el proceso va a ser la clase *LearningMain*, encargada de recoger las opciones seleccionadas por el ususario y ejecutar con los parámetros correctos el algoritmo indicado.

En la figura 3.1 están reflejadas las clases del modelo de análisis. Es importante señalar que en el modelo de diseño se mostrará con más detalle cada una de las clases asociadas a las presentadas aquí pero, como hemos dicho, el modelo de análisis es tan sólo un primer intento por definir los conceptos claves que describen el sistema.

3.1.3. Modelo de diseño

El diseño de los algoritmos de aprendizaje se ha dividido en dos partes: diseño arquitectónico y diseño detallado. El diseño arquitectónico describe la división en subsistemas y las interfaces por las que se comunican, con el objetivo de determinar el marco de referencia que guiará la construcción del sistema. El diseño detallado describe cada subsistema pormenorizadamente, con el objetivo de facilitar la fase de implementación.

Diseño arquitectónico

Nuestro objetivo al diseñar la arquitectura es aislar en paquetes (módulos) distintos los aspectos diferentes del problema para que las sucesivas ampliaciones sean sencillas. Una arquitectura se describe con tres modelos o vistas:

- Vista estática: describe qué componentes tiene la arquitectura.

[2]Para ver en detalle estos algoritmos véase la sección 3.4.

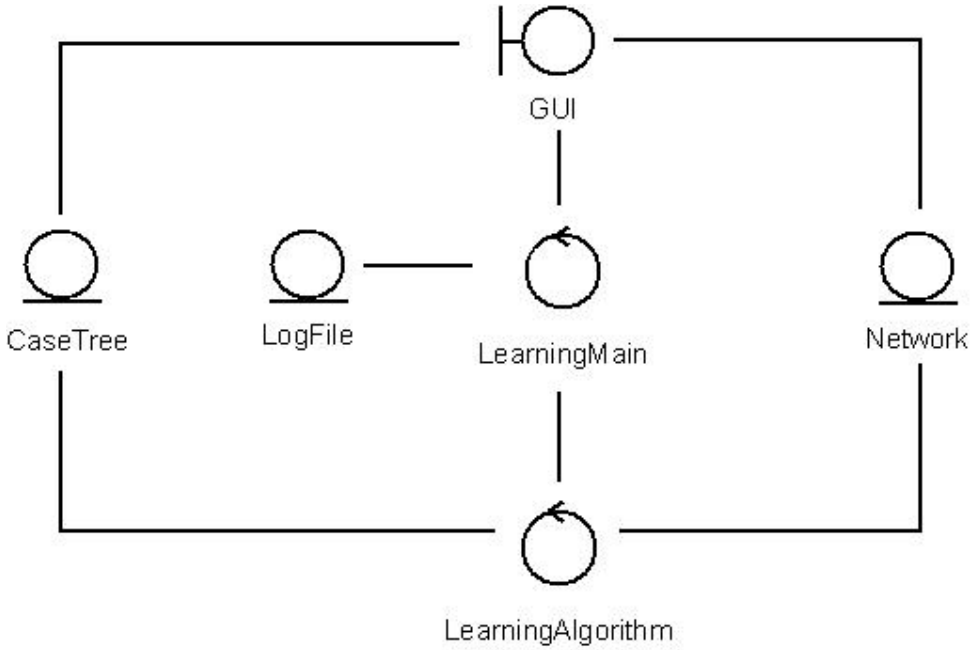


Figura 3.1: Modelo de análisis de un algoritmo de aprendizaje genérico en Carmen.

- Vista funcional: describe qué hace cada componente.
- Vista dinámica: describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí.

En el diseño, una de las soluciones más recomendables, es encontrar un patrón (solución ya probada para un problema específico de diseño) que sea aplicable. Aunque existen numerosos patrones arquitectónicos, ninguno de ellos es adecuado para describir esta arquitectura, así que pasaremos a describir cada vista una por una usando los diagramas usuales del UML.

Vista estática Cada uno de los paquetes es un conjunto de clases que cooperan entre sí con una finalidad común. La definición de cada uno de los paquetes se ha realizado atendiendo a los criterios de cohesión (los elementos de un paquete han de estar relacionados funcionalmente entre sí, es decir, han de realizar una misma función general) y acoplamiento (es conveniente una baja dependencia entre los módulos de un programa para así minimizar la propagación de errores y facilitar la ampliación del sistema). Los paquetes relativos a los algoritmos de aprendizaje son los reflejados en la figura 3.2.

Vista funcional Los algoritmos están codificados en el paquete *learning*, que se compone de los paquetes *metrics, casetree, preprocess y gui*, que se explican a continuación.

El paquete *metrics* codifica las métricas usadas en la evaluación de las redes.

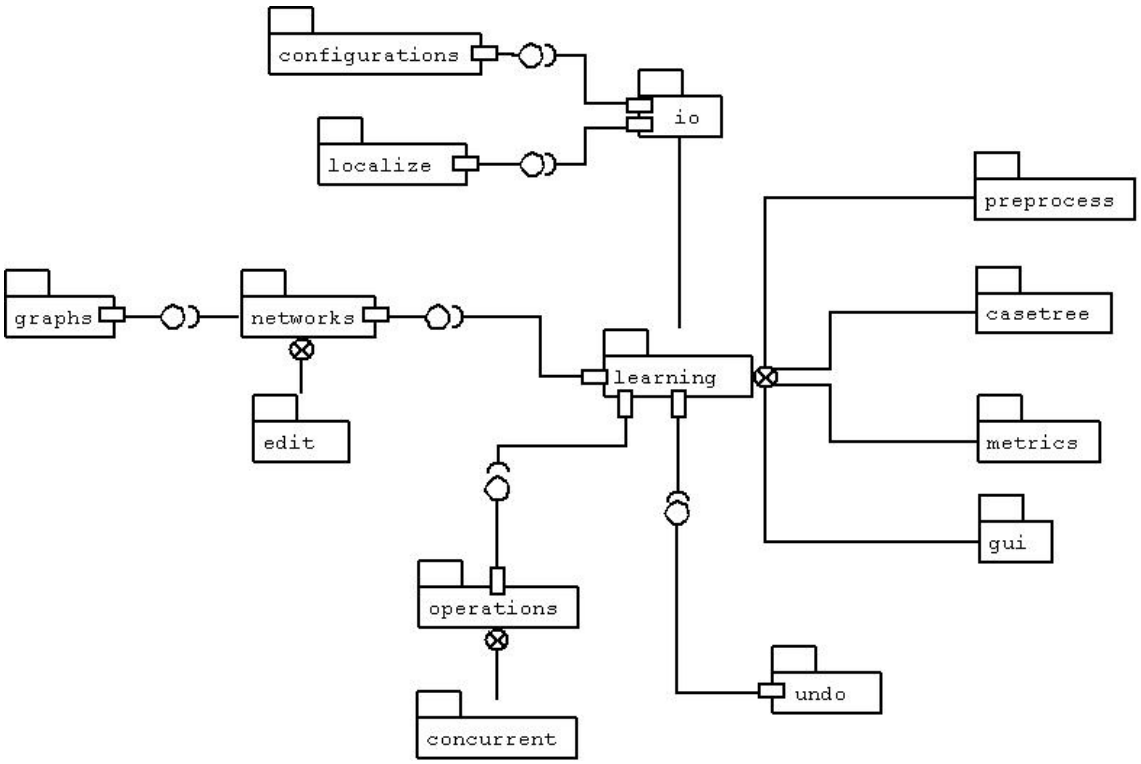


Figura 3.2: Paquetes y sus relaciones. El símbolo $\otimes-$ significa *se compone de*. El símbolo $-O$ significa *interfaz* y el símbolo $-C$ significa *usa interfaz*.

El paquete *casetree* se encarga del almacenamiento y gestión de los casos de la base de datos, permitiendo calcular las frecuencias absolutas de cada una de las configuraciones de las variables.

El paquete *preprocess* se encarga del preprocesamiento de los datos almacenados. Permite dos tipos de preprocesamiento: el tratamiento de valores ausentes en la base de datos y la discretización de variables numéricas.

El paquete *gui* es el encargado de mostrar la interfaz gráfica que se le presenta al usuario para elegir las distintas opciones parametrizables y lanzar el proceso de aprendizaje.

Uno de los paquetes importantes es *operations*, que codifica las operaciones elementales con potenciales (suma, multiplicación, marginalización y división) utilizando el algoritmo de los offsets acumulados.

Un subpaquete de éste es *concurrent*, para operaciones en paralelo, que no ha dado el rendimiento que esperábamos debido a problemas con el modelo de memoria utilizado por la máquina virtual. El problema surge cuando se comparten estructuras de datos de gran tamaño.

El paquete *undo* se encarga de gestionar las operaciones de hacer / deshacer de los algoritmos, que definen cada uno de sus pasos (añadir o eliminar un enlace de la red) como ediciones. Una edición es una acción codificada en una clase y se definen en el paquete *edit*, asociado a la clase *networks*. Describiremos todo esto en detalle más adelante.

El paquete *io* se encarga de la entrada / salida (lee y escribe archivos de modelos gráficos probabilistas). Se compone de otros dos paquetes: *configuration* que sirve para leer de disco un conjunto de variables del sistema y *localize*, que sirve para mostrar las cadenas de caracteres del programa en varios idiomas.

El paquete *networks* codifica los modelos probabilistas que existen hasta el momento (redes bayesianas, diagramas de influencia y otros). Aunque en el caso del módulo de aprendizaje sólo se usa para codificar redes bayesianas.

El paquete *graphs* asociado a *networks* contiene el aspecto gráfico de las redes del paquete anterior. Este paquete es utilizado desde la interfaz gráfica de Carmen para representar las redes aprendidas por pantalla.

Vista dinámica Como cada algoritmo es diferente describiremos esquemáticamete el funcionamiento de un algoritmo genérico.

Hay tres fases: inicio, operación y finalización. Hemos omitido toda la información no esencial porque se expone en la parte dedicada al diseño detallado.

- Fase de inicio: se crean los objetos que necesita el algoritmo y se lanza el mismo.

- Fase de operación: el algoritmo se ejecuta siendo controlado por el paquete *undo* y usando los servicios proporcionados por los paquetes *networks, operations, casetree y metrics*.
- Fase de finalización: el objeto ejecutor tiene los resultados del algoritmo y destruye los objetos creados en la fase de inicio. En general, se aprovecha esta etapa para, una vez construido el grafo, llevar a cabo el aprendizaje paramétrico y devolver así al objeto ejecutor, la red bayesiana completa.

Diseño detallado

Control En la arquitectura, el control es la forma de decidir cuándo y cómo un módulo proporciona sus servicios. Existen dos estilos: centralizado y basado en eventos. El control centralizado supone una organización piramidal de los módulos, donde los que están en la cima controlan a los demás y es propio de la programación estructurada. En el control basado en eventos cada subistema puede responder a eventos generados externamente y es propio de la programación orientada a objetos.

En el control basado en eventos cada componente posee varios fragmentos de código llamados manejadores de eventos. Los manejadores son llamados por un despachador de eventos, que se encarga de decidir qué componente maneja cada evento. Todo este esquema está basado en el concepto de invocación implícita, que consiste en que un componente genera un evento y aquellos componentes que han registrado su interés reciben dicho evento invocándose el manejador asociado.

Como hemos señalado anteriormente, resulta aconsejable hacer uso en la medida de lo posible, de patrones de diseño, puesto que son soluciones ya probadas a problemas que ocurren de un modo recurrente. En este caso, haremos uso de los patrones *Observer* y *Command*. Para cada uno de los patrones, daremos una descripción tal y como aparece en la literatura, para después describir las modificaciones que hemos hecho sobre cada uno de ellos.

Patrón Observer Dado un objeto que puede cambiar de estado y otros objetos que están interesados en ese posible cambio, este patrón define una forma de notificar los cambios. El objetivo de este patrón es desacoplar a los observadores (clientes) del objeto observado. La forma de capturar los eventos de cambio es que cada observador se suscribe a un "listener".

El objeto *ConcreteObservable* contiene una colección de *Observer*s. Cuando ocurre un cambio sobre *ConcreteObservable*, el propio *ConcreteObservable* se encarga de avisar a todos los *Observer*s de dicho cambio usando el método *notifyObservers()*.

La secuencia de eventos que ocurre con este patrón está indicada en la figura 3.4: el objeto observado se inicializa con los objetos interesados en sus cambios y cada vez que hace algo que modifica su estado se envía un mensaje indicando dicha modificación a los observadores que tiene almacenados. Los observadores modifican sus estructuras de datos del modo que sea oportuno.

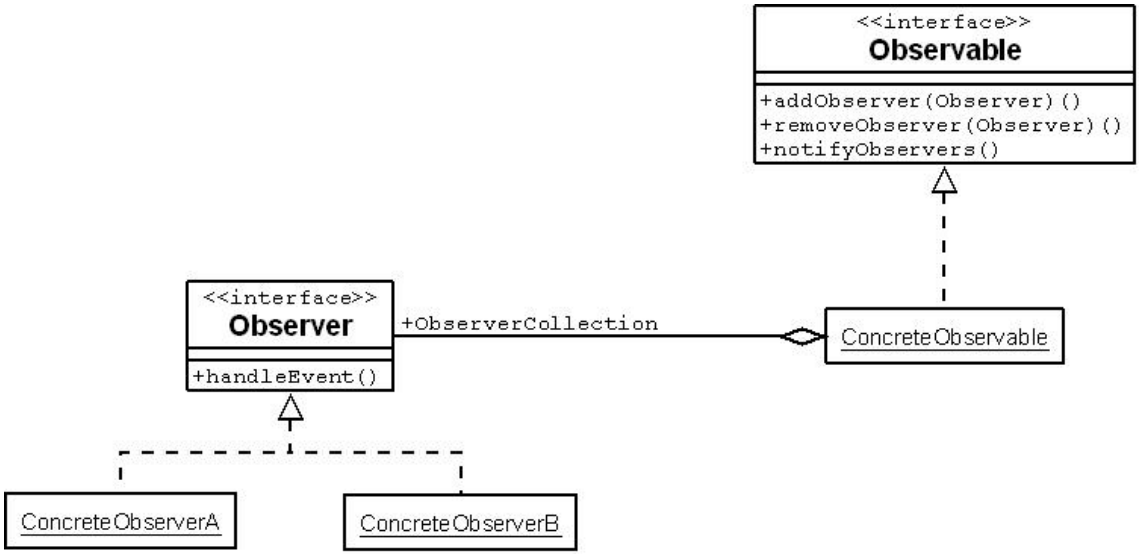


Figura 3.3: Diagrama de clases UML del patrón Observer.

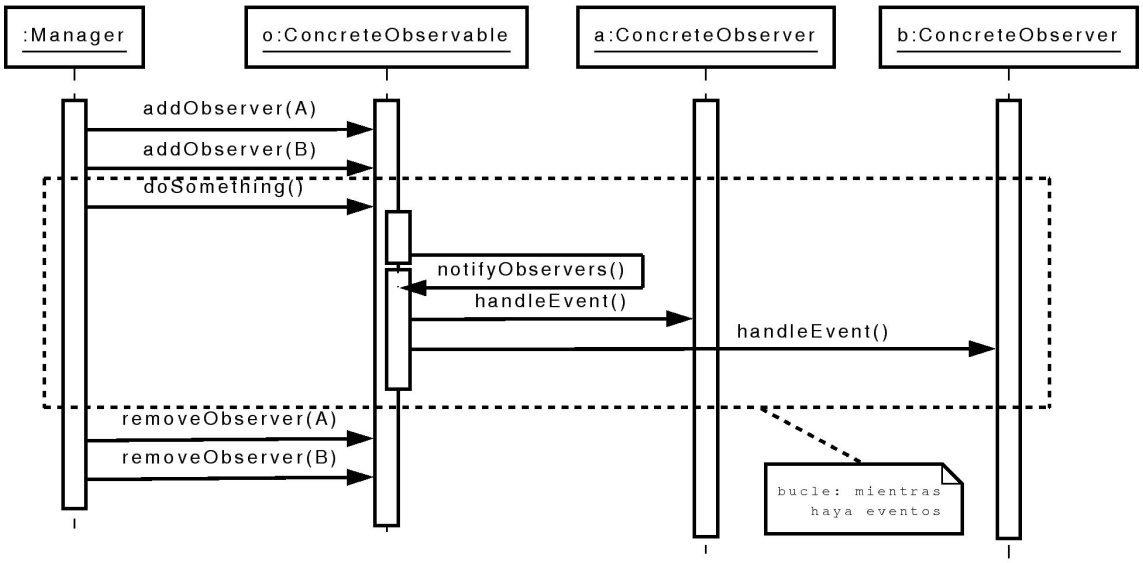


Figura 3.4: Diagrama de secuencias del patrón Observer.

Modificación del patrón Observer Puede darse la circunstancia de que se intente hacer un cambio sobre el objeto observado que no sea "legal" desde el punto de vista de otro objeto. Esto puede ocurrir por algún tipo de error no previsible por parte del programador; un ejemplo es lo que ocurre en la interfaz de usuario cuando un usuario intenta añadir un enlace que crea un ciclo sobre un grafo acíclico. Este caso se puede tratar dividiendo el patrón anterior en dos fases: en una primera fase el objeto observado anuncia su intención de hacer un determinado cambio. Los objetos oyentes pueden vetar ese cambio si no lo consideran correcto en cuyo caso no se realiza. Si no hay veto por parte de ningún objeto oyente el objeto observado actúa como en el caso anterior realizando el cambio y anunciándolo.

En este caso, en el diagrama de clases se modifica la interfaz *Observer* sustituyendo el método *handleEvent* por *announceChange* y *changeHappened*.

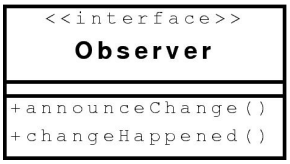


Figura 3.5: Cambios en la interfaz de Observer.

El diagrama de secuencias queda modificado como se indica en la figura 3.6

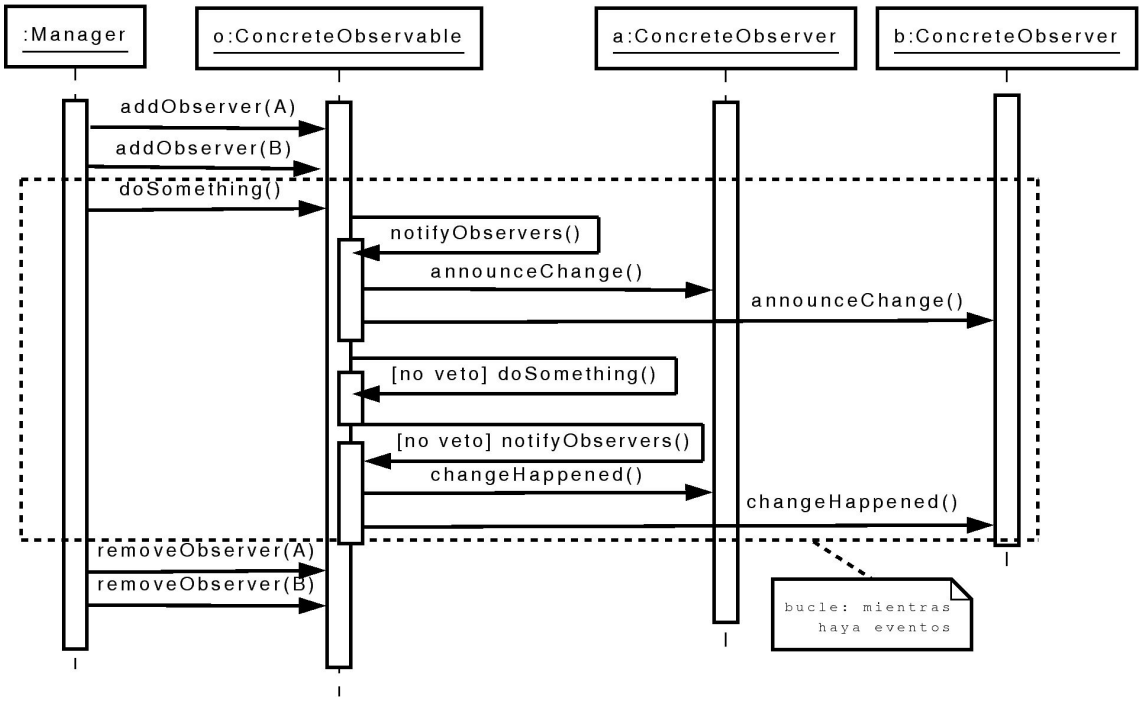


Figura 3.6: Patrón observer de dos fases.

Patrón Command Este patrón encapsula un comando en un objeto de forma que puede ser almacenado, pasado a métodos y devuelto al igual que cualquier otro objeto. Se utiliza generalmente para implementar las acciones de hacer / deshacer, poner comandos en una cola y ejecutarlos en momentos distintos y para desacoplar la fuente de una petición del objeto que la cumple.

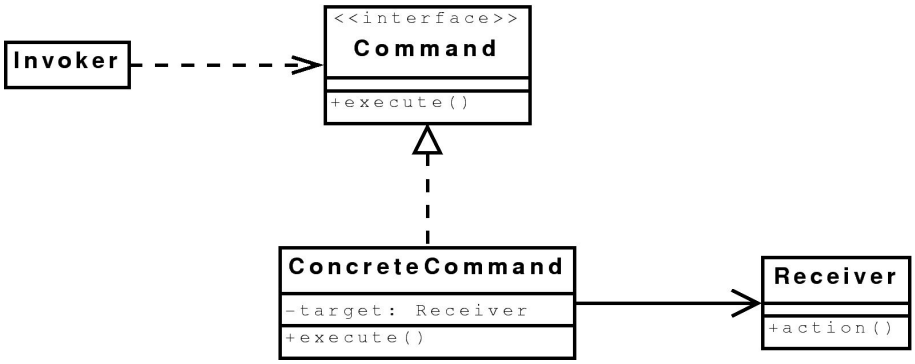


Figura 3.7: Diagrama de clases del patrón Command.

Lo más importante de este patrón es que separa el *cuándo* y el *cómo* en la ejecución de las acciones.

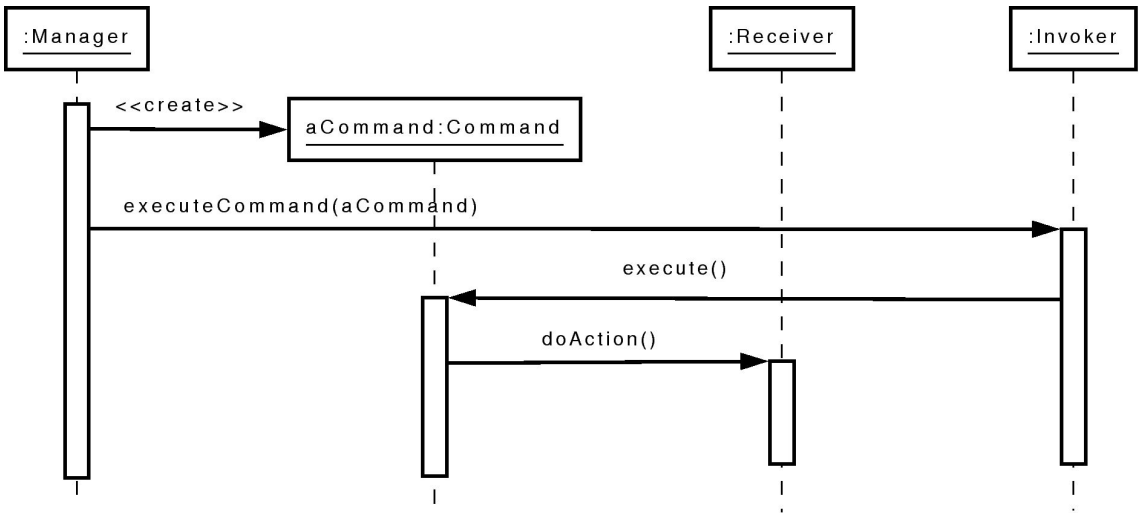


Figura 3.8: Diagrama de secuencias del patrón Command.

Existen dos variaciones del patrón Command. Una es para las acciones de hacer / deshacer y otra para comandos compuestos de otros. Veremos ambas porque las usaremos en el diseño de algoritmos.

Patrón Commnad para hacer/deshacer En este caso es necesario que el objeto *Receiver* de la figura 3.8 sea capaz de invertir la acción realizada, lo cual supone que tiene

que guardar toda la información necesaria. Si se desea poder deshacer varias acciones será necesario guardar las acciones realizadas. Lo más lógico es guardar las acciones en una pila para poder deshacerlas en el orden inverso.

Por otra parte, puede ocurrir que un algoritmo se ejecute todo seguido, sin posibilidad de deshacer los pasos. Este modo de ejecución va a ser más eficiente como mínimo desde el punto de vista de la memoria porque no va a ser necesario que las ediciones guarden información para deshacer.

Patrón Command para acciones compuestas de otras Esta modificación se conoce como *MacroCommand*. Se utiliza en el caso de acciones complejas que se tienen que hacer o todas o ninguna. Su diagrama de clases se muestra en la figura 3.9.

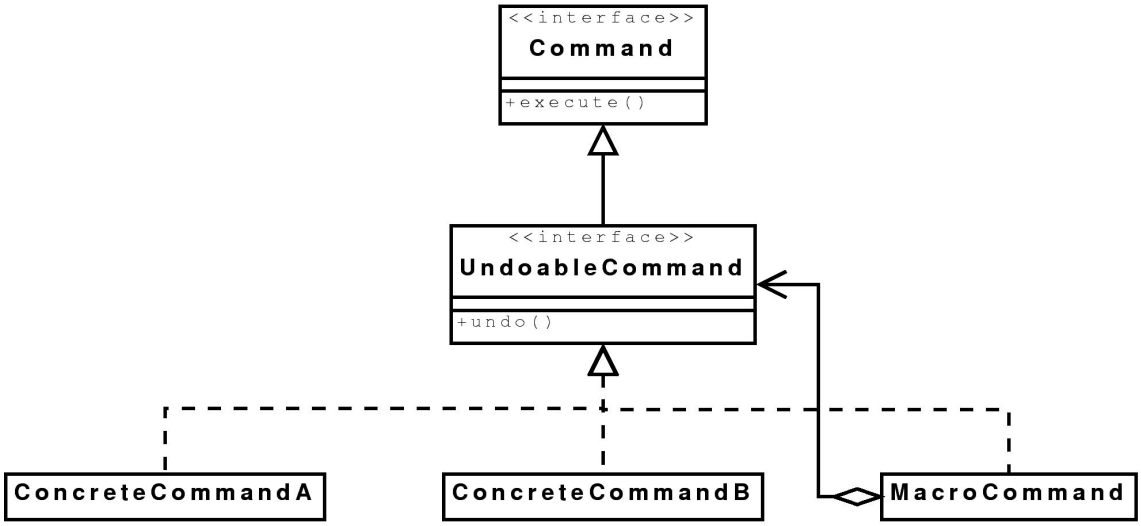


Figura 3.9: Patrón Command con las dos modificaciones para Hacer / Deshacer y acciones compuestas.

3.1.4. Juntando las piezas del modelo de diseño

Para el diseño del marco genérico de los algoritmos de aprendizaje en Carmen hemos combinado el patrón Command para hacer/deshacer (figura 3.9) y el patrón Observer de dos fases (figura 3.6).

De entrada, siguiendo el esquema definido en el patrón Command, un algoritmo se va a dividir en pasos elementales que vamos a llamar *ediciones*. Una edición se codifica como una clase e implementa los métodos *doEdit()* y *undo()* (a los que únicamente hemos cambiado el nombre respecto al patrón Command) definidos en la interfaz *PNEdit (Probabilistic Network Edit)*. Una clase que implemente la interfaz *PNEdit* es responsable de hacer y deshacer la edición que codifique y guarde todos los datos que necesite para ello. La

consecuencia de este esquema desde el punto de vista del programador es que un algoritmo se va a dividir en varias clases, pero cada una de ellas no es más compleja que la parte del algoritmo que implementa.

Por otra parte, y siguiendo el esquema definido en el patrón Observer, en el caso de modelos gráficos probabilistas se trata con estructuras de datos bastante complejas (hay muchas clases de distinto tipo involucradas en la ejecución de un algoritmo). Esta complejidad se ha tratado aislando los diferentes aspectos en diferentes participantes. Los participantes son: el algoritmo, la red probabilista sobre la que opera, la métrica que ha de evaluar cada grafo y el generador de operaciones, que se encarga de dar una lista de operaciones posibles a realizar sobre la red probabilista (dependiendo del algoritmo, estas opciones pueden ser: añadir un enlace –siempre que no genere un ciclo–, eliminar un enlace o invertir un enlace). Cada participante hace una copia de la información que utiliza y se registra en un objeto de la clase *PNESupport (Probabilistic Network Edit Support)*. Cada vez que un participante desea modificar algo se lo indica al objeto de la clase *PNESupport*, que retransmite ese deseo al resto de participantes. Si ninguno se opone se realiza el cambio.

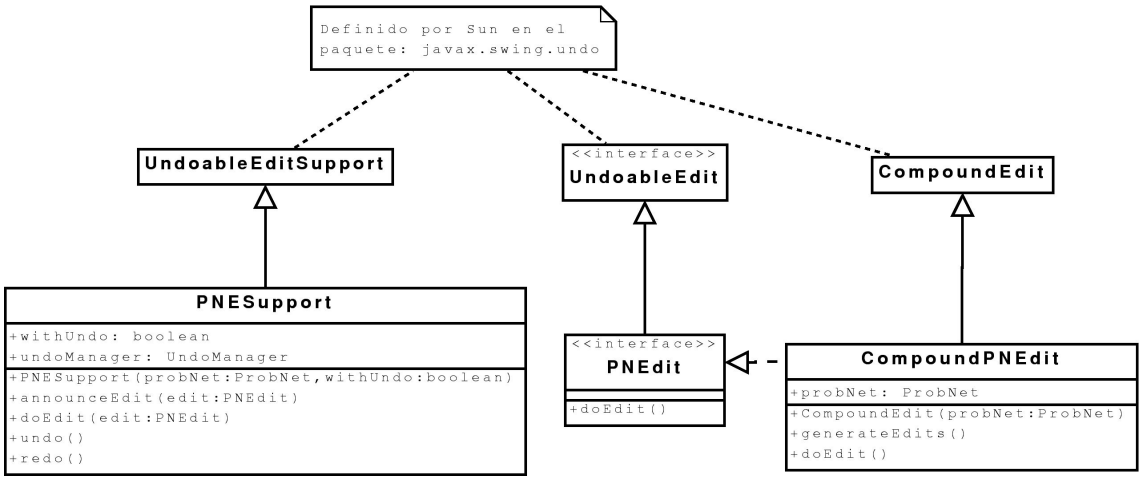


Figura 3.10: Clases e interfaces en el paquete *carmen.undo*.

Algunas aclaraciones sobre el párrafo anterior:

- Existe un único objeto del tipo *PNESupport* al que todos los participantes pueden acceder.
- El objeto *PNESupport* es el que se encarga de gestionar la realización de ediciones.
- Expresar el deseo de realizar una modificación significa que se crea el objeto asociado a la edición correspondiente a la acción y se pide al objeto de tipo *PNESupport* que se encargue de anunciarlo.

El paquete *undo* tiene muchas más clases que las indicadas en la figura 3.10 y las clases tienen más métodos que los indicados. Las clases y los métodos que se han incluido son lo esencial para entender el funcionamiento de las ediciones.

Modelo de diseño de un algoritmo Este diseño lo vamos a expresar desde dos puntos de vista: estático y dinámico. El punto de vista estático representa los componentes de los que está formado y sus relaciones, que se representarán mediante un diagrama de clases. Mientras que, el punto de vista dinámico representa la interacción de esos componentes para realizar una función. En este apartado usaremos un diagrama de secuencias.

Punto de vista estático En el caso de Carmen un algoritmo de aprendizaje va a operar sobre un modelo probabilista. Podemos representar estos modelos por la clase *ProbNet* que es la clase antecesora de todas ellas. Para englobar todos los algoritmos de aprendizaje se ha creado una interfaz *LearningAlgorithm* común a todos ellos con un método llamado *run()*. Ese método *run()* va a devolver el modelo probabilista (*ProbNet*) aprendido . Vamos a suponer que el algoritmo se divide en una sola edición que llamaremos *Edition* y que utilizaremos un objeto de tipo *Metric* y un objeto de tipo *EditionsGenerator* que serán usados por el objeto de tipo *LearningAlgorithm*.

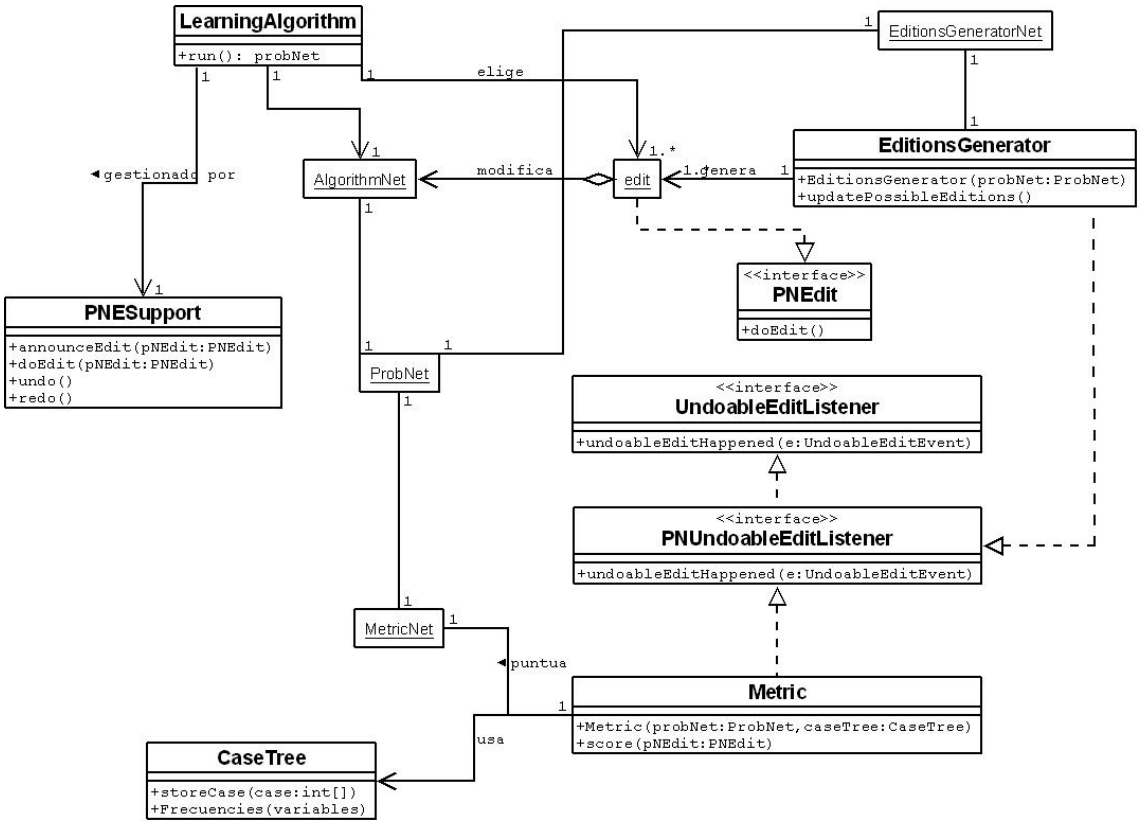


Figura 3.11: Diagrama de clases de un algoritmo genérico.

Cada uno de los diferentes participantes hace una copia de la red probabilista recibida

(*ProbNet*). Cada copia está adaptada a las características del participante: *AlgorithmNet* para *LearningAlgorithm*, *MetricNet* para *Metric* y *EditionsGeneratorNet* para *EditionsGenerator*. Cuando se reciben los eventos (ediciones) generados por el algoritmo las copias son actualizadas. En el diagrama de clases de la figura 3.11 es la clase *LearningAlgoritm* la que va a actuar como objeto observado y las clases *Métrica* y *Generador de operaciones* como observadores. Por ese motivo la clase *LearningAlgorithm* es gestionada por la clase *PNESupport* y las clases *Métrica* y *EditionsGenerator* implementan la interfaz *PNUndoableEditListener* para escuchar los eventos de cambio y actualizar sus copias locales.

El motivo de que cada participante tenga su propia copia es que cada participante puede haber adaptado la red a las características específicas del proceso que lleva a cabo. Así, se facilita la capacidad de ampliación de la herramienta Carmen, uno de los principales requerimientos señalados durante la etapa de análisis de requisitos. El único inconveniente es el mayor consumo de memoria, que es despreciable porque lo que realmente ocupa espacio son las tablas de probabilidades que en ningún caso es necesario duplicar.

Punto de vista dinámico Ahora veremos cómo es el esquema anterior durante la ejecución del algoritmo con el diagrama de secuencias de la figura 3.12.

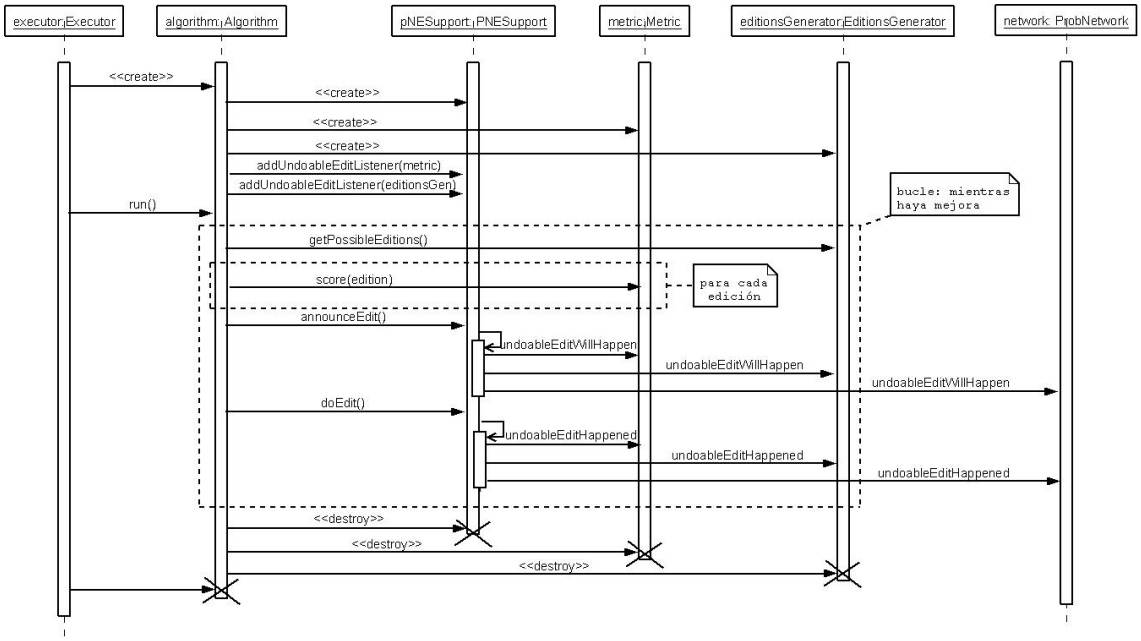


Figura 3.12: Diagrama de secuencias simplificado del funcionamiento de un algoritmo.

Un algoritmo es iniciado por algún objeto, por ejemplo la interfaz gráfica. La ejecución de un algoritmo tiene lugar en tres fases: lo primero que hace un objeto algoritmo es crear los objetos necesarios para su funcionamiento; por ejemplo: una métrica y un generador de ediciones, un objeto del tipo *PNESupport* si no existe y una copia de la red probabilista sobre la que opera. Además de crear dichos objetos, aquellos objetos interesados en las operaciones del algoritmo se registran en el objeto *PNESupport*. La segunda fase es la de

operación: en general, en cada iteración del bucle principal del algoritmo de aprendizaje, el generador de operaciones ha de crear la lista de posibles operaciones que se pueden realizar sobre la red con la que se está trabajando para que posteriormente, el algoritmo pida a la métrica que evalúe las redes generadas al aplicar cada una de esas operaciones por separado. Finalmente el algoritmo ha de seleccionar una de dichas redes y confirmar la acción que la ha generado. Para ello, le comunica al objeto *PNESupport* la operacion *PNEdit* realizada, y éste se encarga de comprobar si hay algún veto. Si no lo hay se envía un mensaje a *PNESupport* para que realize la acción. Una vez terminado el aprendizaje estructural, ha de realizarse el aprendizaje paramétrico, que consiste en construir para cada nodo el potenical de frecuencias absolutas normalizado, a partir de las configuraciones de todos sus padres, teniendo en cuenta la corrección de Laplace. Por último, en la fase de finalización se devuelve la red obtenida a la interfaz gráfica para ser representada en pantalla.

Por último, se muestran a continuación, como ayuda a los desarrolladores, los pasos que hay que seguir para implementar un nuevo algoritmo de aprendizaje en Carmen:

1. Crear una clase dentro del paquete *carmen.learning.algorithms* que herede de la clase abstracta *LearningAlgorithm*.

2. Implementar el método *run()* atendiendo al diagrama de secuencias mostrado en la figura 3.12.

3. Añadir una nueva entrada al array de *String algorithms* de la clase *LearningMain* para que el nuevo algoritmo de aprendizaje sea accesible desde la interfaz gráfica, y añadir un nuevo caso en la selección de algoritmo en esa misma clase.

4. Es posible que, en algunos casos, sea necesario modificar la clase *editionsGenerator*, que da la lista de ediciones (añadir o eliminar un enlace) posibles sobre la red con la que se está trabajando. Aunque, en general, los algoritmos de aprendizaje suelen trabajar en cada paso con la vecindad formada por la adición o eliminación de un enlace.

3.2. Ejemplo: Algoritmo del gradiente

En este punto vamos a ver una realización concreta de todas las ideas de la sección 3.1 aplicada al algoritmo del gradiente. Dado que el diseño arquitectónico coincide con el comentado en la sección anterior, comentaremos tan sólo el diseño detallado del algoritmo. Siguiendo el esquema de la sección 3.1 veremos los diagramas de clases que ilustran el aspecto estático del problema y a continuación comentaremos su aspecto dinámico.

Punto de vista estático Como ya hemos comentado, el método principal de todo algoritmo de aprendizaje es el método run(). Basta llamar a este método para obtener la red buscada.

La métrica es cualquier objeto que implemente la interfaz *Metric*, que dada una red, calcula la puntuación de dicha red con una nueva edición a través del método score(:PNEdit).

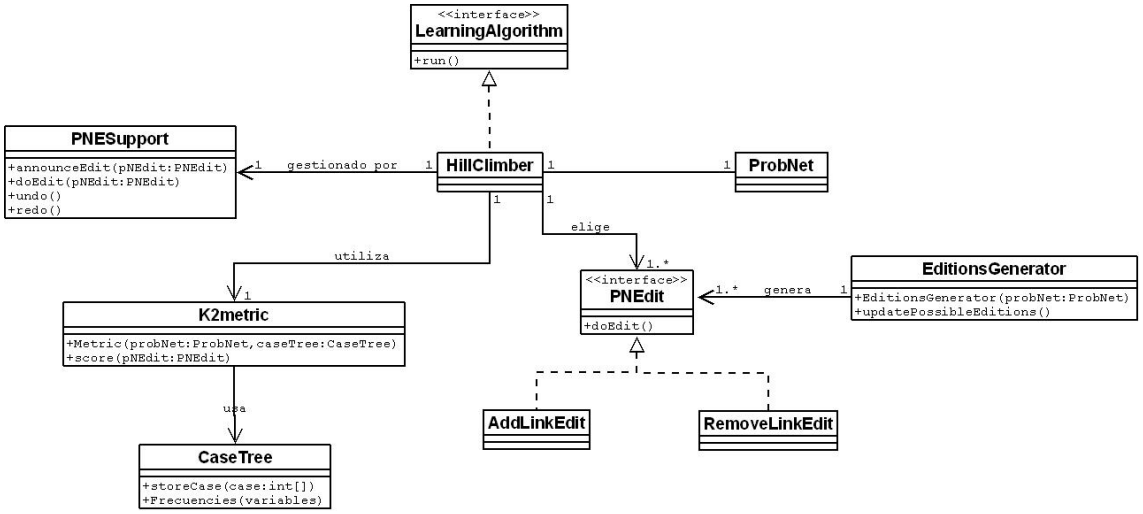


Figura 3.13: Diagrama de clases del algoritmo del gradiente.

La métrica es independiente del algoritmo usado, pero con el objetivo de ilustrar este ejemplo usaremos la métrica K2, implementada en la clase *K2metric*.

La función del objeto *PNESupport* es controlar las ediciones que operan sobre la red, que pueden ser añadir o quitar un enlace. Para este algoritmo en concreto no se definen objetos que veten la adición o eliminación de un enlace y en consecuencia después de anunciar cada edición, ésta se lleva a cabo.

El diagrama de clases de la figura 3.13 muestra las clases involucradas en el proceso de aprendizaje, aunque se han omitido la mayor parte de los atributos y de los métodos de las clases para no complicar demasiado la figura.

Punto de vista dinámico Vamos a suponer que el algoritmo es lanzado por un objeto llamado *Lanzador*, que puede ser la interfaz gráfica, una clase de pruebas, etc. El ejemplo del algoritmo del gradiente se ajusta perfectamente al diagrama de secuencias mostrado en la figura 3.12. La única salvedad es que, tras puntuar cada una de las redes resultantes de aplicar las distintas operaciones generadas, el algoritmo del gradiente escoge la edición con mejor puntuación siempre y cuando se mejore la puntuación de la red actual.

Como hemos indicado antes, el algoritmo puede dividirse en dos etapas: la etapa de lanzamiento y la de operación (en este caso en la etapa de finalización no es necesario realizar ninguna acción). En la etapa de lanzamiento, el algoritmo crea los objetos de tipo *EditionsGenerator* y *Metric* y los registra en el objeto de la clase *PNESupport*. A continuación, el algoritmo pasa a la etapa de operación, cuyo funcionamiento es el mostrado en el pseudocódigo del algoritmo 4, en la página 25. Una vez terminada la ejecución, ya con el grafo construido, el algoritmo realiza el aprendizaje paramétrico para obtener las tabalas de probabilidad asociadas a cada nodo de la red y devuelve la red probabilista completa.

3.3. Métricas

Una métrica no es más que una función que asigna a cada red bayesiana un valor que indica su calidad teniendo en cuenta un conjunto de datos dado. Como vimos en la sección 2.5, las métricas más utilizadas pueden dividirse en tres grupos: métricas bayesianas, métricas de longitud mínima de descripción y medidas de información. Esta clasificación se ha tenido muy presente a la hora de diseñar la estructura de clases del paquete *metrics*.

A continuación, se comentan las cuestiones de diseño relativas a las métricas de calidad. Dada la menor complejidad de este elemento del módulo de aprendizaje, esta sección es mucho más breve que la anterior, y se centra en el análisis de requisitos y en el punto de vista estático del modelo de diseño.

3.3.1. Análisis de requisitos

Como es lógico, los requisitos que ha de cumplir cualquiera de las métricas implementadas son los requisitos de la herramienta Carmen en general y del módulo de aprendizaje en particular. Sin embargo, en el diseño e implementación de las métricas se ha prestado especial atención a dos requisitos:

- Extensibilidad: Dado que las métricas de calidad son uno de los puntos del programa que pueden ser, muy probablemente, objetivo de ampliación (es muy probable que alguien quiera experimentar con distintas métricas de todo tipo) es necesario prestar una especial atención a este requisito en la etapa de diseño.
- Eficiencia: En los algoritmos de aprendizaje mediante búsqueda heurística, el punto más conflictivo en cuanto a consumo de tiempo son las sucesivas llamadas a la métrica de calidad, de ahí que sea necesario optimizar al máximo los cálculos realizados para obtener la puntuación final de cada una de las redes.

3.3.2. Modelo de diseño

Como ya hemos comentado antes, dada la mayor simplicidad de las métricas de calidad, en este punto vamos a centrarnos tan sólo en el punto de vista estático, mostrando los distintos componentes y sus relaciones mediante un diagrama de clases. El punto de vista dinámico no es necesario comentarlo puesto que el funcionamiento de una métrica es muy sencillo y ya ha sido mostrado en la sección 3.1: cada vez que se realiza una edición sobre la red, se puntúa la nueva red.

Punto de vista estático El diagrama de clases relativo a las métricas se muestra en la figura 3.14. Para englobar todas las métricas de calidad se ha creado una clase abstracta *Metric* con un método llamado *score()* que ha de ser implementado por todas sus subclases y que debe devolver la puntuación de la red. Para cumplir con el requisito de eficiencia, se ha de evitar realizar todos los cálculos cada vez que se llama a este método. Como las sucesivas redes con las que se va a trabajar se generan añadiendo, eliminando o invirtiendo un único enlace, basta con realizar los cálculos a los que afecta ese nuevo enlace y obtener la calidad de la nueva red a partir del valor de calidad de la anterior.

Por otra parte, como se señaló en el diseño de los algoritmos de aprendizaje, las métricas han de recibir los eventos que se generen sobre la red que se está construyendo para actualizar la copia que poseen de ella. De ahí que la clase *Metric* implemente la interfaz *PNUndoableEditListener*.

De la clase *Metric* heredan directamente dos clases principales: *BayesianMetric* y *EntropyMetric*, que se corresponden con dos de los tres tipos de métricas estudiados en la sección 2.5. El hecho de que no se incluya directamente una subclase que agrupe a las métricas de longitud mínima de descripción se debe a que éstas están estrechamente relacionadas con las medidas de información[3] (o de entropía) de modo que todas ellas pueden ser englobadas por la clase *EntropyMetric*.

Cada uno de estos dos subtipos tienen características en común que hacen aconsejable esta estructura de clases. Las métricas bayesianas comparten las funciones de cálculo global de la métrica, variando tan sólo los calculos de la puntuación de cada uno de sus nodos al añadirse o eliminarse un enlace. Por su parte, las métricas de longitud mínima de descripción y las de entropía tienen en común los cálculos de la entropía y la dimensión de un nodo (aunque este último no es utilizado por todas las métricas) cuando se le añade o elimina un enlace.

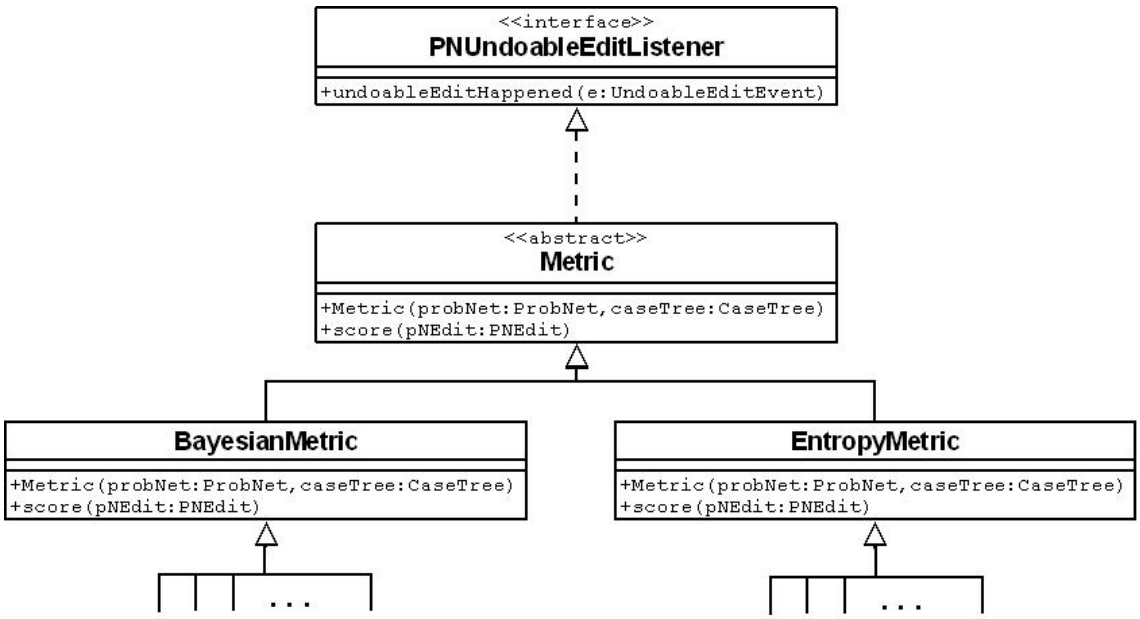


Figura 3.14: Diagrama de clases del paquete *metrics*.

Por último, se muestran a continuación, como ayuda a los desarrolladores, los pasos que hay que seguir para implementar una nueva métrica de aprendizaje en Carmen:

[3]La relación entre las medidas de longitud mínima de descripción y las medidas de información queda patente a la vista de las ecuaciones 2.25 y 2.26, (pág 22).

1. Crear una clase dentro del paquete *carmen.learning.metrics*. En principio, una métrica puede ser cualquier clase que herede de la clase abstracta *Metric*, pero es recomendable que la herencia se haga a través de las clases *BayesianMetric* o *EntropyMetric* dependiendo del tipo de métrica que se quiera implementar. En caso de que se quiera desarrollar un nuevo tipo de métrica (distinto de las métricas bayesianas, de longitud mínima de descripción y de entropía), es conveniente realizar una clase que herede de la clase *Metric* para englobar a todas las métricas de ese tipo.

2. Implementar el método *score(PNEdition)* que devuelve la puntuación de una red dada. En caso de estar implementando una métrica bayesiana (es decir, una subclase de la clase *BayesianMetric*) basta con reimplementar los métodos *nodeScore* y *nodeScoreRemovedLink*, que calculan, respectivamente, la puntuación de un nodo que recibe un nuevo enlace o que se le ha eliminado uno de los enlaces que recibía. En caso de estar implementando una subclase de la clase *EntropyMetric* basta con reimplementar los métodos *score()* asociados a cada tipo de edición.

3. Añadir una nueva entrada al array de *String metrics* de la clase *LearningMain* para que la nueva métrica de calidad sea accesible desde la interfaz gráfica, y añadir un nuevo caso en la selección de métrica en esa misma clase.

3.4. Almacenamiento de casos

En esta sección se explica el almacenamiento en memoria de los casos leídos y el algoritmo utilizado para calcular las frecuencias absolutas de cada una de las posibles configuraciones de las variables.

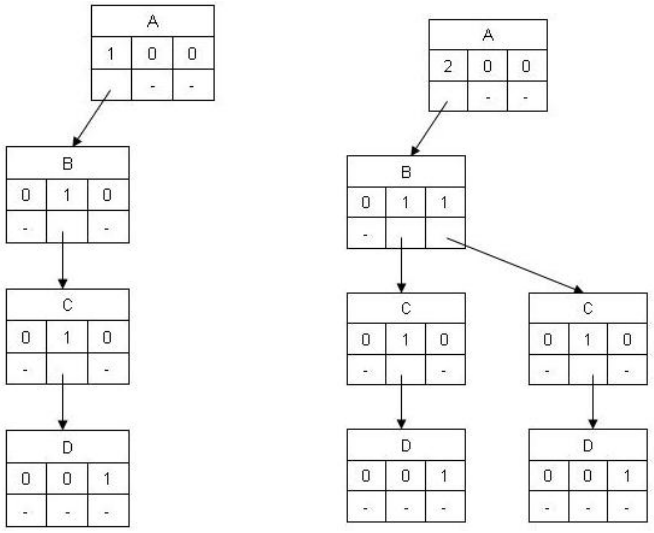
3.4.1. Árbol de casos

Para almacenar los casos en memoria se ha utilizado una estrucutra de árbol en la que cada nodo tiene una variable asociada, una lista de hijos con tantos elementos como valores pueda tomar dicha variable y una lista de contadores que indican el número de casos que existen en la base de datos con cada uno de los valores de la variable asociada.

El árbol de casos se construye dinámicamente al leer la base de datos. Cuando se lee un caso, se desciende en el árbol comenzando a partir del nodo raíz. Si no existe ningún hijo asociado al valor que se ha leído para la variable asociada al nodo, se crea ese nuevo hijo y se incrementa su contador, en caso contrario simplemente incrementamos el contador y repetimos el proceso desde el hijo asociado a dicho valor (de esta forma evitamos crear nodos de manera innecesaria. Si tenemos tres variables con 1000 posibles valores cada una pero en la base de datos sólo aparece una única configuración tendremos un árbol con tan sólo 3 nodos, mientras que el árbol completo tendría 1.000.001 nodos!!).

Véamos el funcionamiento del algoritmo con un ejemplo: supongamos que tenemos una base de datos con cuatro variables (A, B, C y D) con tres valores cada una (0, 1 y 2). Supongamos que el primer caso que nos encontramos en la base de datos es el [0, 1, 1, 2]. Entonces, el aspecto del árbol tras almacenar ese primer caso sería el mostrado en la figura 3.15(a). Al ver que la variable 'A' toma el valor '0', se crea un hijo asociado a ese

valor y se incrementa el contador asociado a ese valor. A continuación se baja al siguiente nivel del árbol y se repite el proceso teniendo en cuenta ahora que la variable 'B' toma el valor '1'. Así se desciende en el árbol hasta almacenar por completo el caso.



(a) Estado del árbol tras almacenar el caso (0, 1, 1, 2).

(b) Estado del árbol tras almacenar el caso (0, 2, 1, 2).

Figura 3.15: Evolución de la creación de un árbol de casos. En cada nodo, la primera fila indica la variable asociada, la segunda contiene la frecuencia de cada uno de los estados de dicha variable y la tercera contiene los punteros a los hijos asociados a cada estado.

Si a continuación queremos almacenar el ejemplo [0, 2, 1, 2], como ya existe un nodo asociado al valor '0' de la variable 'A', basta con incrementar el contador asociado a dicho valor y descender en el árbol. A continuación, vemos que la variable 'B' toma el valor '2' y, al no existir un hijo asociado a ese valor, hay que crearlo. A continuación se repite el proceso para las variables 'C' y 'D'. El árbol obtenido tras almacenar este segundo ejemplo es el mostrado en la figura 3.15(b).

Como hemos podido comprobar, el uso de esta estructura de árbol minimiza la memoria ocupada por los ejemplos de la base de datos y, como veremos a continuación, permite calcular las frecuencias totales de las distintas configuraciones de variables de una forma muy eficiente.

3.4.2. Algoritmo de cálculo de frecuencias absolutas

El cálculo de las frecuencias absolutas de cada una de las configuraciones de una lista de variables dada requiere el uso de un array de offsets que permita movernos en la tabla de frecuencias absolutas. El array de offsets debe contener en cada una de las casillas asociadas a las variables de interés (que son aquellas cuyas frecuencias absolutas queremos conocer), el número de configuraciones posibles de sus predecesores en la lista de dichas variables. El cómputo de ese array es muy sencillo, veámoslo con un ejemplo:

Supongamos que tenemos 5 variables binarias: A, B, C, D y E y que las variables de interés son C, A y D. El array de offsets tendrá 4 elementos (4 es el mayor de los índices de las variables de interés). Para construir el array de offsets comenzamos por asignar un 1 al elemento asociado a la primera de las variables de interés (en nuestro caso es la variable C, por tanto pondremos un 1 en la tercera casilla del array de offsets, puesto que el índice de la variable C es 3). A continuación, a cada una de las restantes variables de interés le asignaremos el producto del último valor insertado en la tabla multiplicado por el número de estados de la última variable con la que hemos trabajado. De este modo, el array de offsets contiene, para cada variable de interés, el número de configuraciones posibles de sus predecesores en la lista de variables de interes (como hemos visto, en el caso de la primera variable de la lista, que no tiene predecesores, se inserta un 1). Así, en nuestro ejemplo, el array de offsets resultante sería: [2, 0, 1, 4]. Como ya hemos dicho, estos valores nos indican cómo movernos en la tabla de frecuencias absolutas. Por ejemplo (véase la figura 3.16) , el primer elemento de la tabla de frecuencias absolutas indica la frecuencia de los casos con C = 0, A = 0 y D = 0. Si queremos obtener la frecuencia de los casos con C = 0, A = 1 y D = 0 tendremos que sumar 2 (el offset indicado en el array calculado previamente para la variable A) y movernos hasta el tercer elemento del array. Si queremos consultar la frecuencia de los casos con C = 0, A = 1, y D = 1, tendremos que sumar al valor anterior el offset de la variable D, que es la que ha cambiado, indicándonos que debemos mirar en la casilla 6 para obtener la frecuencia absoluta de dicha configuración.

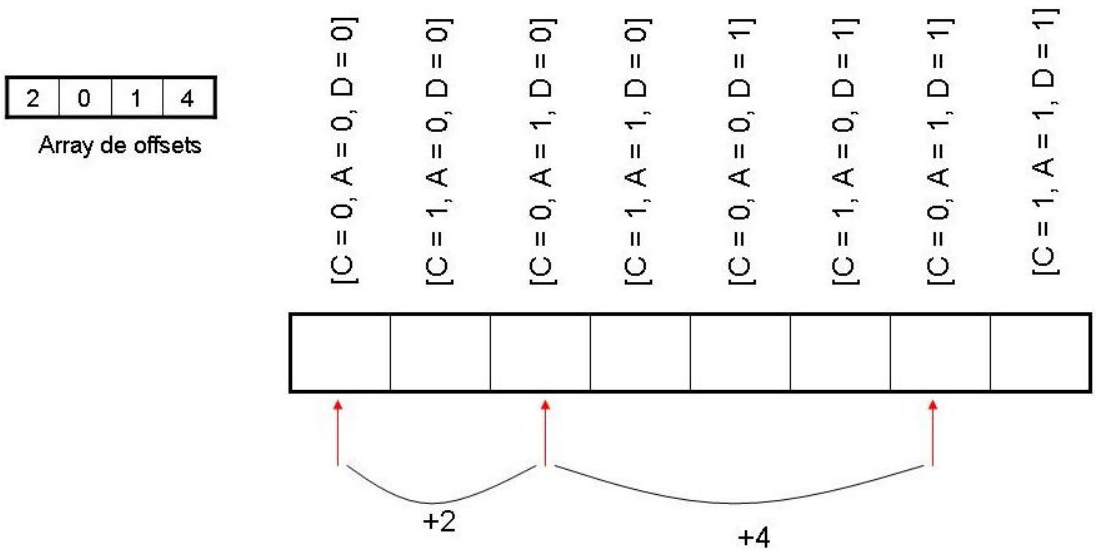


Figura 3.16: Búsqueda de valores en la tabla de frecuencias absolutas.

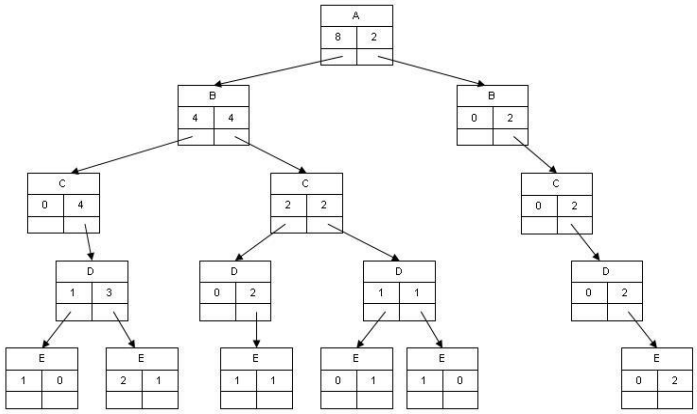
Una vez construido el array de offsets se recorre el árbol de casos en profundidad para obtener la tabla de frecuencias absolutas buscada. Para ello se desciende en el árbol recursivamente actualizando correctamente el valor del elemento de la tabla que queremos rellenar (haciendo uso del array de offsets creado y teniendo en cuenta que cada nivel del árbol corresponde a una variable, de modo que bajar un nivel en el árbol implica sumar el

offset asociado a esa variable) hasta alcanzar el nivel de la variable de interés con mayor índice. Al llegar a dicho nivel basta con sumar los valores de la tabla de contadores que posee cada nodo a los elementos correspondientes en la tabla de frecuencias absolutas. El pseudocódigo del algoritmo 5 muestra su esquema de funcionamiento. Continuando con el ejemplo anterior y observando la figura 3.17 podemos entender fácilmente el funcionamiento del algoritmo. Supongamos que el árbol construido a partir de la base de datos es el mostrado en la figura 3.17(a). El algoritmo comienza a descender por la rama asociada al valor '0' de la variable A hasta llegar al primer nodo de la variable C. Como el valor '0' de dicho nodo no tiene hijos, pasamos al siguiente valor sumando el offset de la variable con la que estamos trabajando, en este caso 1 (figura 3.17(b)). A continuación, descendemos un nivel más en el árbol y, como hemos alcanzado el nivel máximo de profundidad (recuérdese que la variable de interés de mayor índice era la variable D), actualizamos la tabla de frecuencias absolutas. En la posición en la que nos encontramos sumamos el contador asociado al primero de los valores de la variable D y, a continuación, nos desplazamos en la tabla sumando el offset asociado a la variable D y sumamos el contador asociado al segundo de los valores (figura 3.17(c)). Una vez que se han sumado los contadores asociados a cada valor de la variable D, se sube en el árbol (véase que al subir un nivel en el árbol, el offset recupera el valor que tenía en ese nivel) hasta alcanzar la siguiente rama que no ha sido recorrida aún (en nuestro caso, la rama asociada al valor 1 de la variable B) y se repite el proceso hasta haber recorrido todas las ramas.

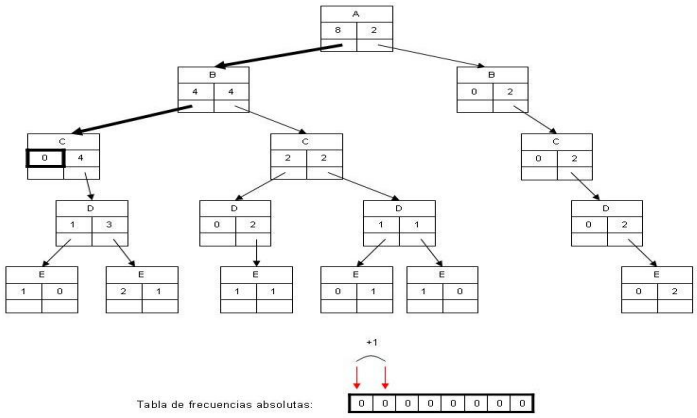
Algoritmo 5: Cálculo de frecuencias absolutas de las posibles configuraciones

Input: nivel actual, máximo nivel a alcanzar, offset inicial, tabla de frecuencias, array de offsets
Result: tabla con las frecuencias absolutas de cada configuración

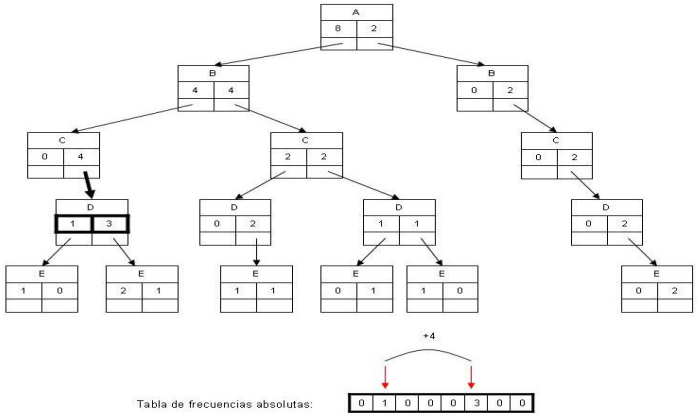
```
1  offsethijo ← 0 //Offset para moverse en el array de contadores de cada nodo
2  offsetvariable ← 0 //Offset para moverse en la tabla de frecuencias.
3  foreach hijo do
4      //Condición de parada
5      if nivel actual es igual al máximo nivel then
6          //Actualizar la tabla de frecuencias y los offsets
7          tabladefrecuencias[offsetinicial] ←
           tabladefrecuencias[offsetinicial] + contadordecasos[offsethijo]
8          offsetinicial ← offsetinicial + offsets[nivelactual]
9          offsethijo++
10     else
11         if el hijo tiene casos asociados then
12             //Llamada recursiva: bajamos al siguiente nivel en el árbol
13             cuenta casos(nivel actual + 1, máximo nivel, offset inicial + offset
               variable, tabla de frecuencias, array de offsets)
14         //Pasamos al siguiente hijo
15         offsetvariable ← offsetvariable + offsets[nivelactual]
```



(a) Árbol a partir del cual queremos calcular las frecuencias absolutas.



(b) Rama sin hijos: se continúa por la siguiente rama actualizando el offset.



(c) Máxima profundidad alcanzada: se actualiza la tabla de frecuencias.

Figura 3.17: Evolución del algoritmo de cálculo de frecuencias absolutas.

3.5. Interfaz de usuario

Las distintas funcionalidades del módulo de aprendizaje se ofrecen al usuario a través de una interfaz gráfica que ha sido implementada usando la herramienta *NetBeans*[4] y diseñada atendiendo a una serie de criterios que se exponen a continuación:

- Consistencia: La consistencia de un aplicación se basa en seguir un comportamiento homogéneo en las entradas y salidas del sistema (uniformidad en la entrada de los datos, en la presentación de la información . . .). El propósito básico de la consistencia es permitir al usuario generalizar el conocimiento acerca de uno o varios aspectos del sistema.
- Familiaridad: La presentación debe ser semejante a otras aplicaciones para que los usuarios que ya tienen experiencia con otros programas similares puedan aprovechar sus conocimientos. En este sentido, se ha tenido muy en cuenta la interfaz gráfica del programa Elvira.
- Claridad: La información ha de ser presentada de una forma clara, concisa y fácilmente accesible. Es necesario mostrar toda la información importante, pero sin llegar a ofrecer más información de la necesaria.
- Facilidad de uso: La interfaz gráfica debe estar diseñada de modo que una persona que no esté familiarizada con el uso de este tipo de interfaces sea capaz de manejarla tras unas breves explicaciones.
- Minimización de las posibilidades de error: Proveer al usuario solamente los comandos que son posibles ejecutarse bajo ciertas circunstancias es una buena manera de prevenir errores. En este sentido, Carmen muestra deshabilitadas aquellas opciones que no se permiten bajo determinadas circunstancias o limita el rango de algunos parámetros para evitar los posibles errores cometidos por un usuario inexperto.
- Flexibilidad: La interfaz debe permitir seleccionar las opciones en distinto orden, permitiendo al usuario moverse libremente por las distintas pantallas.
- Realimentación: Es conveniente que el resultado de cualquier operación realizada por el usuario tenga un efecto visible en la interfaz, de modo que el usuario pueda comprobar que las opciones elegidas son correctas, evitando así posibles errores.
- Robustez: La interfaz gráfica ha de estar preparada para manejar errores inesperados como errores de acceso a disco, mostrando un mensaje descriptivo del error que ha ocurrido y sus posibles soluciones.
- Compatibilidad: Es conveniente que la interfaz facilite la compatibilidad con otros programas similares. En este sentido, Carmen permite abrir bases de datos en los formatos utilizados por *Elvira* y *Weka* además del formato propio de Carmen[5], permitiendo transformar ficheros de un formato a otro facilitando así la comparación entre las distintas herramientas.

[4] `http://www.netbeans.org`

[5] Una descripción de los distintos formatos aceptados por el módulo de aprendizaje de Carmen puede verse en la sección A.8.

El resultado del proceso de diseño e implementación es una interfaz amigable que permite al usuario interactuar con el programa de forma sencilla y eficiente. La apariencia gráfica de la interfaz y el modo de utilizarla puede consultarse en el apéndice A de este mismo documento.

Capítulo 4

Aplicaciones

Es una verdad muy cierta que, cuando no esté a nuestro alcance determinar lo que es verdad, deberemos seguir lo que es más probable.

Descartes, "Discurso del Método".

A partir de 1990, el número de aplicaciones de los modelos gráficos probabilistas (MGPs) ha crecido de manera espectacular dando lugar a un amplio abanico de ámbitos en los que el uso de los MGPs en general y las redes bayesianas en particular ha resultado muy útil. En este apartado nos centraremos en analizar los resultados obtenidos por el módulo de aprendizaje desarrollado, tanto en el ámbito de la medicina (usando una base de datos de mortalidad en enfermos de cáncer del Hospital Universitario 12 de Octubre de Madrid y una base de datos de mortalidad en la UCI polivalente del Hospital Universitario Arnau de Vilanova de Lleida) como en el ámbito económico (usando una base de datos de Caja Madrid sobre impagos de hipotecas).

En las siguientes secciones se realiza un análisis de cada una de las tres bases de datos utilizadas y se aplican algunas de las técnicas de aprendizaje implementadas, con el fin de mostrar las capacidades y limitaciones del módulo de aprendizaje desarrollado. Además, en la etapa de análisis de las bases de datos, se indican las opciones de preprocesado utilizadas para las distintas variables. El módulo de aprendizaje implementado permite realizar dos tipos de preprocesado, uno encaminado al tratamiento de datos ausentes y otro dirigido a la discretización de las variables numéricas. En cuanto al tratamiento de los valores ausentes, se permite mantener dichos valores (añadiendo por tanto un nuevo estado a cada variable con un valor no especificado) o eliminar aquellos registros de la base de datos que tengan una de las variables seleccionadas para el aprendizaje sin especificar. Como se señala en la sección A.5 del manual de usuario, es conveniente ser muy cuidadoso al seleccionar esta opción de tratamiento de valores ausentes, puesto que si la base de datos presenta gran cantidad de valores ausentes, la elección de esta opción puede reducir el número de casos para el aprendizaje hasta valores demasiado pequeños, rebajándose así enormemente la calidad del aprendizaje. Por tanto, antes de elegir esta opción, es recomendable tener

cierto conocimiento de la base de datos para evitar sesgar el aprendizaje de una manera muy importante. Por último, en cuanto a las opciones de discretización, Carmen permite discretizar variables numéricas en intervalos de igual frecuencia o igual anchura (además de permitir la discretización manual). Es importante resaltar que la discretización de variables numéricas es muy recomendable, puesto que en caso de no discretizarlas, dichas variables tendrán un número de estados demasiado elevado (un estado para cada uno de los valores que aparezcan en la base de datos), lo cual puede llevar a una pérdida de eficiencia muy importante.

Una vez finalizado el proceso de aprendizaje, se muestran las redes obtenidas. A día de hoy, Carmen nos permite ver las correlaciones existentes entre las distintas variables involucradas en el proceso de aprendizaje, pero no permite conocer el sentido ni la intensidad de dichas correlaciones. Para obtener este tipo de información que, como veremos más adelante, puede resultar muy importante, se ha utilizado el programa Elvira[1], que sí ofrece esta funcionalidad. Elvira ofrece una idea aproximada de la información numérica contenida. Esto hace que cada enlace se muestre con un color y una anchura propios. Hablando grosso modo, los enlaces coloreados en rojo indican que hay una influencia positiva, es decir que al aumentar el valor del nodo padre aumenta el valor del nodo hijo; los enlaces en azul indican influencia negativa; los enlaces negros indican que el enlace no transmite información y los enlaces en violeta indican que hay influencia ambigua, es decir, que para ciertos valores del padre hay influencia positiva y para otros negativa. Cuanto mayor es la influencia de un nodo sobre otro, mayor es la anchura del enlace.

A la hora de interpretar los enlaces de la red y los colores de cada uno, hay que tener en cuenta cómo están ordenados los estados de cada una de las variables. Por ejemplo, si tenemos un enlace rojo que une el atributo 'altura' y el atributo 'edad', no quiere decir que a mayor altura, mayor edad, sino que a mayor índice de los estados del atributo 'altura' se dan los estados de mayor índice del atributo 'edad'. De este modo, si los estados del atributo 'edad' son: 50, 30, 10, y los estados del atributo 'altura' son: 1, 2 y 3, lo que nos indica el enlace rojo es que una mayor altura se da en las edades menores (los estados del atributo 'altura' con mayor índice se corresponden con los de mayor índice del atributo 'edad'). Por otra parte, en el caso de variables categóricas también se hace necesario conocer la ordenación de los estados para interpretar correctamente los colores de los enlaces. Así, si tenemos un enlace rojo entre las variables 'sexo' y 'edad', para saber si las mayores edades corresponden a hombres o a mujeres, debemos mirar la ordenación de los estados de la variable 'sexo' (si los estados son: 'F' y 'M', las mayores edades corresponden a los hombres ('M')).

Por otra parte, es muy importante tener en cuenta que las redes bayesianas aprendidas sólo pueden ser interpretadas en sentido probabilista, nunca en sentido causal. Aunque, como señalamos en la sección 2.6, existen algunas técnicas que permiten obtener conclusiones causales, éstas no han sido implementadas en este módulo.

[1] `http://www.ia.uned.es/~fjdiez/bayes/elvira`

4.1. Aplicaciones en medicina

La medicina es el campo donde más aplicaciones de MGPs se han construido. Ya desde la década de los 60 se comenzaron a desarrollar sistemas de diagnóstico basados en técnicas bayesianas. Desde entonces, y más aún desde los años 80 cuando se desarrollaron las redes bayesianas y los diagramas de influencia, el crecimiento del número y variedad de aplicaciones en este campo ha sido impresionante. Los modelos gráficos probabilistas y sus métodos asociados son especialmente adecuados para trabajar con incertidumbre, de ahí que el alto número de fuentes de incertidumbre que se encuentra en el campo de la medicina, unido al conocimiento causal, correspondiente a los mecanismos patofisiológicos, explican el gran número de aplicaciones destinadas a manejar problemas de diagnosis (que consisten en construir una hipótesis sobre la enfermedad que sufre el paciente), prognosis (que consisten en hacer una predicción acerca de lo que pasará en el futuro), selección de tratamiento (consistentes en elegir el tratamiento óptimo) o descubrimiento de interacciones funcionales (a partir de redes aprendidas automáticamente, se pueden descubrir nuevas interacciones entre las variables del problema).

Así pues, la principal dificultad que presenta el ámbito de la medicina es la gran cantidad de incertidumbre que se ha de manejar. Esta incertidumbre puede proceder de numerosas fuentes que podemos agrupar en:

- Información incompleta: La información disponible acerca del paciente suele ser incompleta, no sólo en cuanto a su historial médico sino también en cuanto a los síntomas padecidos.
- Información errónea: Puede producirse una descripción incorrecta de los síntomas, errores en el historial clínico del paciente o errores en las pruebas de laboratorio.
- Información imprecisa: Existen numerosas variables difíciles de cuantificar como por ejemplo el dolor, de ahí que las descripciones de los síntomas del paciente e incluso las observaciones del propio médico pueden ser imprecisas.
- Mundo real no determinista: En muchas ocasiones, las mismas causas producen efectos distintos en pacientes distintos e incluso pueden producirse variaciones aleatorias.
- Modelo incompleto: En medicina existen muchos fenómenos cuyas causas se desconocen o sobre las que no existe un acuerdo generalizado y, aunque dicha información estuviera disponible, el número de variables con las que se puede trabajar en un tiempo aceptable es limitado, de modo que los modelos con los que se trabaja siempre suelen ser incompletos.
- Modelo inexacto: Para cuantificar la incertidumbre son necesarios una serie de parámetros que, rara vez, estarán disponibles. Por tanto, estos parámetros han de ser estimados, con el error que dicha estimación puede conllevar.

4.1.1. Mortalidad en enfermos de cáncer de pulmón

La primera de las bases de datos contiene información de 2992 enfermos de cáncer de pulmón ingresados en el Hospital Universitario 12 de Octubre de Madrid entre Septiembre

de 1993 y Septiembre de 1997.

Análisis de la base de datos

La base de datos contiene 31 variables, algunas de las cuales fueron desechadas inicialmente por no aportar información relevante (por ejemplo, la variable 'codifili' es una cadena de caracteres que identifica de forma única a cada uno de los pacientes de la base de datos o la variable 'fechater', que indica la fecha de registro en la base de datos). Para acotar aún más el número de variables utilizadas en el aprendizaje (recordemos que en aprendizaje automático, la dimensionalidad del problema es un factor a tener muy en cuenta para mejorar los resultados tanto en términos de eficacia como de eficiencia) se recurrió a la ayuda de un experto. El Doctor D. Agustín Gómez de la Cámara, Jefe de la Unidad de Investigación del Hospital Universitario 12 de Octubre de Madrid, nos indicó las variables que podrían resultar más relevantes para el problema. Las variables seleccionadas fueron[2]:

- *pleuvisp*: afectación pleural visceral en muestra histológica.
- *pared_p*: afectación de la pared torácica en muestra histológica.
- *pleupa_p*: afectación pleural parietal en muestra histológica.
- *edad*.
- *sexo*.
- *imc*: Índice de Masa Corporal.
- *tamatum_*: tamaño del tumor.
- *EstadoSup*: indica si el paciente murió o sobrevivió.

Una vez seleccionadas las variables que se van a tener en cuenta en el proceso de aprendizaje, es conveniente aplicar distintas técnicas de preprocesamiento para facilitar el aprendizaje y mejorar sus resultados. En cuanto al tratamiento de valores ausentes, se optó por la opción de eliminar aquellos registros en los que alguna de las variables seleccionadas tenía un valor desconocido. Se consideró que ésta era la opción más conveniente teniendo en cuenta que la densidad de la base de datos era bastante alta, de modo que la eliminación de algunos de sus registros no conllevaría graves problemas, de hecho, tras la eliminación, quedaron 2751 registros, lo cual supone una pérdida no demasiado importante. Por otra parte, en cuanto a la discretización, se optó por las siguientes opciones[3]:

- *edad*: Se discretizó en 3 intervalos de igual anchura, para adecuarse aproximadamente a las etapas de joven, adulto y anciano.
- *imc*: Se discretizó en 3 intervalos de igual anchura: bajo, normal y elevado.

[2]En caso de no tener acceso al conocimiento de un experto sobre el tema tratado, puede ser conveniente aplicar distintas técnicas de selección de variables con el objetivo de elegir aquellas que sean más relevantes para el problema en cuestión.

[3]Las variables que no aparecen en esta lista no son numéricas y, por tanto, no son discretizables.

- *tamatum_*: Se discretizó en 4 intervalos de igual anchura atendiendo a las indicaciones del experto.
- *pared_p*: No se discretizó, puesto que tan sólo tiene tres valores: 0, 1 y 2.

Resultados obtenidos

Con las opciones indicadas en el apartado anterior, la red obtenida tras el proceso de aprendizaje se muestra en la figura 4.1. Esta red nos permite ver las correlaciones existentes entre las distintas variables involucradas en el proceso de aprendizaje. Como hemos comentado anteriormente, puede resultar muy útil conocer el sentido y la intensidad de la correlación e incluso comprobar si alguno de los enlaces obtenidos es un enlace espurio. Para ello hemos utilizado el programa Elvira. Así, guardando la red obtenida en Carmen en formato Elvira, y abriendo dicha red desde Elvira, obtenemos la red de la figura 4.2.

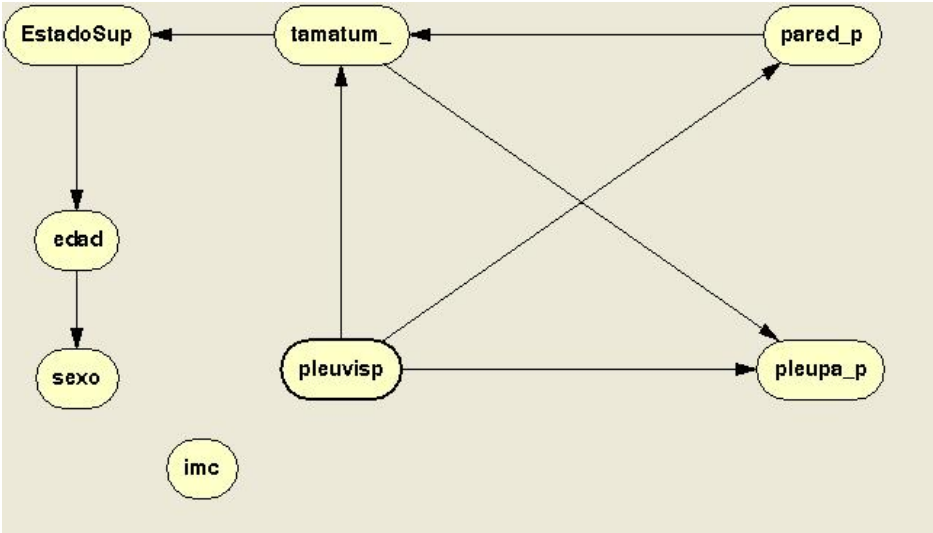


Figura 4.1: Red aprendida a partir de la base de datos de mortalidad en enfermos de cáncer.

Como hemos comentado, la red obtenida ha de ser interpretada en sentido probabilista, es decir, podemos extraer conclusiones acerca de las correlaciones existentes entre las distintas variables. De este modo, a partir de los resultados obtenidos podemos obtener relaciones que, a priori, podrían no ser obvias. Por ejemplo, el enlace existente entre las variables 'sexo' y 'edad' indica que el cáncer de pulmón afecta a hombres y mujeres a distintas edades. Si observamos la ordenación de los estados de las dos variables ('sexo': 'M', 'F' y 'edad' 0, 1, 2 representando tres intervalos de discretización en orden creciente) podemos concluir que, dado que existe un enlace rojo entre ambas variables (es decir, hay una correlación positiva), las mujeres suelen desarrollar cáncer de pulmón en edades más avanzadas que los hombres (quizás porque, al menos hace años, las mujeres fumaban menos que los hombres, lo cual es un factor determinante para desarrollar antes cáncer de pulmón. Véase además que en la base de datos hay 2988 casos correspondientes a hombres y tan sólo 219 correspondientes a mujeres).

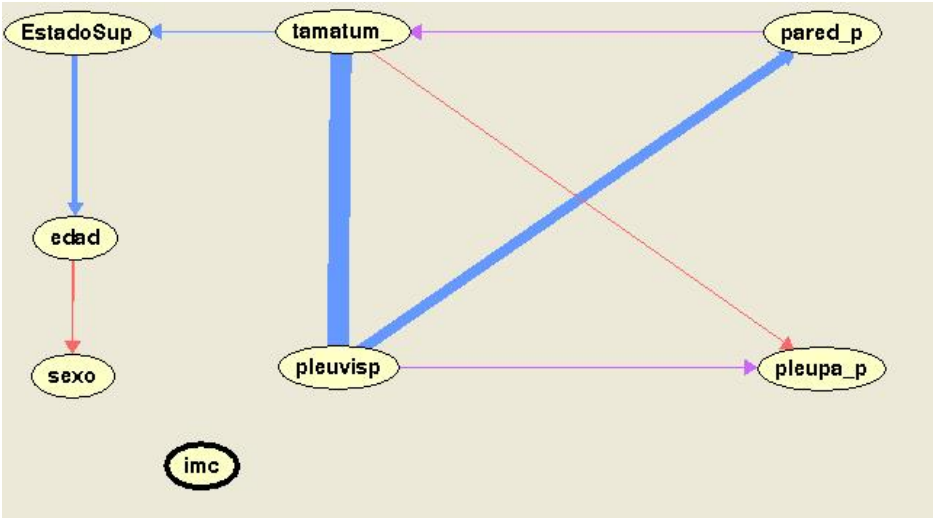


Figura 4.2: Red de mortalidad en enfermos de cáncer, abierta en Elvira.

Por otra parte, podemos confirmar o rechazar hipótesis sobre las relaciones entre las variables. Por ejemplo, en principio, podríamos pensar que el tamaño del tumor en el momento de diagnosticar el cáncer tiene cierta importanica en el resultado final de supervivencia del paciente. A la vista de los resultados obtenidos, podemos confirmar efectivamente esa hipótesis, puesto que existe correlación entre el tamaño del tumor y la supervivencia del paciente.

Además, a partir de las redes obtenidas, podemos extraer información acerca de las relaciones de independencia entre las distintas variables. Por ejemplo, entre las variables 'sexo' y 'estadoSup' existe correlación. Sin embargo, esas variables son condicionalmente independientes dada la edad. Este hecho muestra otra de las posibilidades de uso del módulo de aprendizaje: puede servir para encontrar variables 'intermedias' que expliquen ciertas correlaciones que podrían parecer inexplicables. Supongamos que, en cierta ciudad se ha observado cierta correlación entre el número de cigüeñas y el número de nacimientos. A priori, dicha correlación parece inexplicable (a menos que pensemos que las cigüeñas traen los niños de París). Sin embargo, si aplicamos el módulo de aprendizaje a una base de datos que contenga otras variables que no hayan sido tenidas en cuenta, como por ejemplo el número de habitantes y el número de iglesias de la ciudad, es probable que obtengamos una red similar a la de la figura 4.3. A partir de esa figura, podemos ver que, aunque efectivamente el número de nacimientos y el número de cigüeñas están correlacionados, son variables independientes dado el número de habitantes y el número de iglesias. Por tanto, otra de las posibles utilidades del módulo de aprendizaje implementado es la capacidad de descubrir variables que hasta el momento no habían sido tenidas en cuenta y que pueden explicar ciertas correlaciones que parecían inexplicables.

Así, como hemos visto, el módulo de aprendizaje puede resultar muy útil para encontrar correlaciones, que podrían no ser obvias, entre las distintas variables y, sin embargo, podrían ser de gran utilidad para incrementar el conocimiento sobre algunas enfermedades,

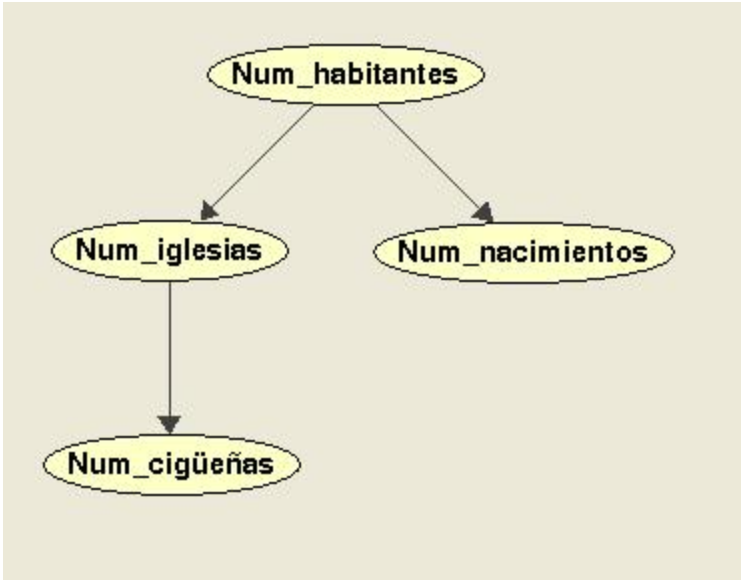


Figura 4.3: Red aprendida que sirve para explicar la correlación nacimientos - cigüeñas.

y ayudar en algunos problemas de diagnosis, prognosis etc. Además de relaciones de dependencia, se puede obtener información muy valiosa sobre las relaciones de independencia condicional presentes entre las variables y, del mismo modo, también se puede utilizar el módulo para confirmar o descartar hipótesis sobre las relaciones de las variables de la red. Además, el módulo puede servir para encontrar variables que no habían sido tenidas en cuenta y que pueden explicar algunas de las correlaciones existentes.

4.1.2. Mortalidad en enfermos ingresados en la UCI

La segunda de las bases de datos que se ha utilizado para probar el módulo de aprendizaje desarrollado contiene información de 2684 pacientes ingresados en la UCI polivalente del Hospital Universitario Arnau de Vilanova de Lleida entre Enero de 1996 y Diciembre del 2006. En este estudio se incluyeron todos los pacientes mayores de 16 años y con estancia en la UCI mayor de 24 horas.

Análisis de la base de datos

El análisis de la base de datos es similar al realizado en el apartado anterior. La base de datos cuenta con 2684 registros de 15 variables, ninguna de las cuales presenta valores ausentes. En este caso, en el proceso de aprendizaje se usaron todas las variables presentes:

- EDAT: Edad en años (mayores de 16 años).
- TAM: Peor tensión arterial media en las primeras 24 horas de ingreso en la UCI. Una TAM más baja implicará peor pronóstico y se asocia con la necesidad de inotropos o el desarrollo de fracaso renal o coagulopatía.
- INOTROPO: Indica si se precisa medicación con inotropos para mantener hemodinámica.

- GLASGO: Escala de Glasgow (rango de 3 a 15). Un valor de 15 indica un estado normal mientras que el valor 3 indica muerte encefálica.
- FIPO2: Fracción inspirada de oxígeno. Se obtiene dividiendo presión parcial de oxígeno en sangre arterial por la fracción de oxígeno aportada (desde 0,21 a 1).
- ACVA: Accidente cerebro-vascular agudo. Hemorragia o isquemia cerebral.
- MASAINTR: Efecto de masa intracraneal.
- IRENAL: Desarrollo de insuficiencia renal aguda.
- INFEC: Proceso infeccioso.
- QUICK: Presencia de coagulopatía.
- PROGRAMA: Ingreso programado en la UCI.
- IORG: Paciente con enfermedad crónica previa que limita sus posibilidades de sobrevivir al ingreso en la UCI.
- VM: Ventilación mecánica.
- DIAG: Diagnóstico (dividido en 8 diagnósticos con evoluciones diferentes).
- MUERTO: Indica si el paciente sobrevivió o no al ingreso hospitalario.

En cuanto a las opciones de preprocesamiento utilizadas, dado que no existen valores ausentes, no fue necesario tomar ninguna decísión en torno a ese tema. Por su parte, se optó por las siguientes opciones de discretización[4]:

- EDAT: Se discretizó en 3 intervalos de igual anchura.
- TAM: Se discretizó en 3 intervalos de igual anchura: bajo, normal y elevado.
- GLASGO: Dado que el número de estados de esta variable es de 13 (muy por debajo del número de estados de una típica variable numérica) finalmente se optó por mantener la variable sin discretizar.
- FIPO2: Se discretizó en 4 intervalos de igual anchura.

Resultados obtenidos

Con las opciones indicadas en el apartado anterior, la red obtenida tras el proceso de aprendizaje se muestra en la figura 4.4. Esta red nos permite ver las correlaciones existentes entre las distintas variables involucradas en el proceso de aprendizaje. Como en los casos anteriores, se ha usado Elvira para obtener más información acerca de las correlaciones encontradas. La red abierta en Elvira presenta el aspecto mostrado en la figura 4.5.

[4]Las variables que no aparecen en esta lista no son numéricas y, por tanto, no son discretizables.

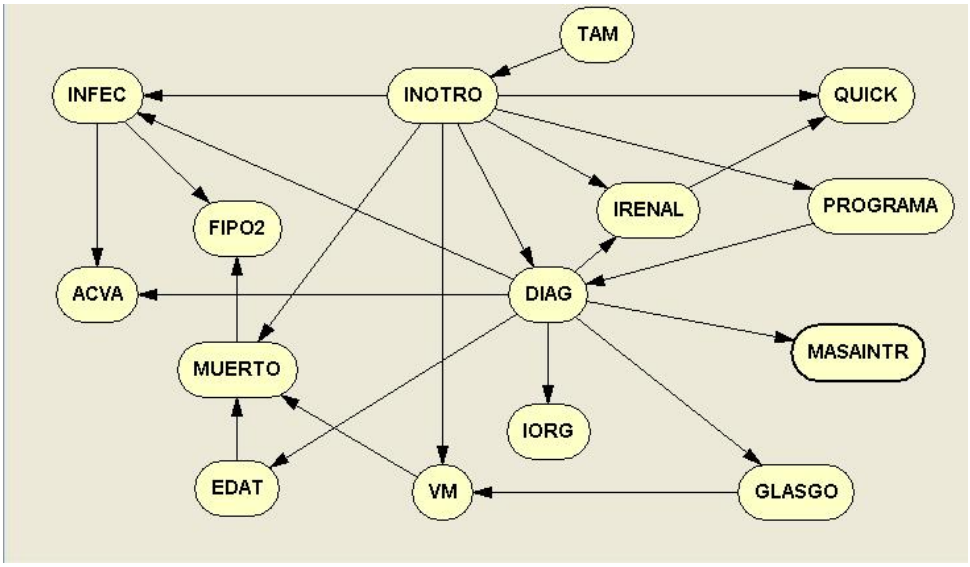


Figura 4.4: Red aprendida a partir de la base de datos de pacientes ingresados en la UCI.

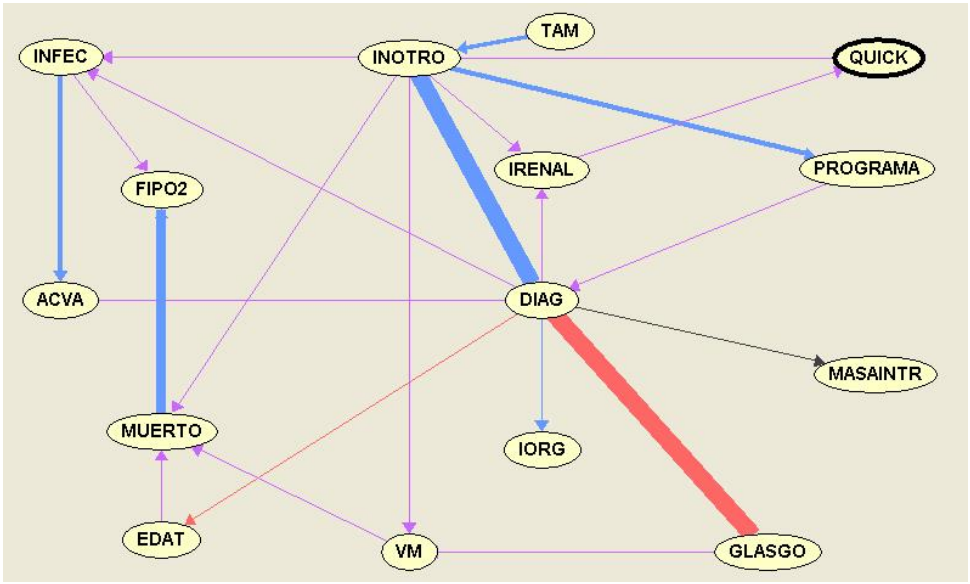


Figura 4.5: Red de pacientes ingresados en la UCI, abierta en Elvira.

En este caso, en primer lugar podemos observar una serie de relaciones que vienen a confirmar algunas de las cosas que ya sabíamos por nuestro conocimiento teórico del problema. Como hemos comentado antes, el módulo de aprendizaje, además de utilizarse para descubrir correlaciones entre las distintas variables, puede utilizarse para confirmar o rechazar hipótesis teóricas sobre dichas relaciones. Por ejemplo, ya hemos comentado que una tensión arterial media (TAM) baja se asocia con la necesidad de inotropos, lo cual queda patente en la red obtenida a la vista del enlace azul que une las variables TAM e INOTROPO (recuérdese una vez más que el color azul indica una correlación negativa entre los índices de los estados). Otra de las hipotésis que se confirman es la relación entre la fracción inspirada de oxígeno y la gravedad del paciente. En la red podemos ver que hay una fuerte correlación entre valores bajos de la variable 'FIPO2' y un resultado final de muerte del paciente.

Por otra parte, podemos observar algunas relaciones que resultan, a priori, bastante obvias: Existe una correlación muy fuerte entre el índice de Glasgow y el diagnóstico del paciente. Valores bajos del índice de Glasgow se corresponden con valores bajos del diagnóstico, lo cual resulta bastante evidente puesto que, cuanto menor es el índice de Glasgow más grave es la situación del paciente, y los valores bajos del diagnóstico se corresponden con diagnósticos más graves. Además, existe una débil correlación entre la edad del paciente y su diagnóstico. Como era de esperar, a mayor edad, peor diagnóstico. Sin embargo, resulta sorprendente el hecho de que dicha correlación, que a priori podría resultar obvia, sea muy débil. Este hecho puede deberse a que, en muchos casos, los ingresos de gente joven se deban a causas mucho más graves (como por ejemplo accidentes de circulación), de modo que la intensidad de la correlación se vería suavizada.

Finalmente, a la vista de la red obtenida, podemos comentar algunas relaciones que podrían no ser tan obvias, al menos para gente no muy experta en estos temas. Por ejemplo, dado el enlace entre las variables INFEC y ACVA, parece que la presencia de un proceso infeccioso favorece un accidente cerebro-vascular agudo (o viceversa, recuérdese que las redes obtenidas no muestran información causal) o que el tratamiento con inotropos está bastante relacionado con un ingreso programado en la UCI. Además, llama la atención la fuerte correlación entre el tratamiento con inotropos y la gravedad del diagnóstico. Este hecho puede explicarse porque el tratamiento con inotropos se asocia a un fracaso hemodinámico, lo cual hace aumentar enormemente la gravedad del diagnóstico.

4.2. Aplicaciones en economía

En el ámbito de la economía las redes bayesianas se han utilizado en numerosas ramas como la toma de decisiones sobre capital de riesgo, marketing, filtrado cooperativo o el análisis de riesgos en la concesión de créditos o hipotecas, que es donde se encuadra el ejemplo que comentamos a continuación. El lector interesado en éstas y otras aplicaciones de las redes bayesianas en este ámbito puede encontrar una amplia descripción con ejemplos reales en el libro de Neapolitan [26], en el que en la primera parte (caps. 1 a 6) se ofrece una excelente revisión de las redes bayesianas y los modelos gráficos probabilistas, en la segunda (caps. 7 a 10) se analizan diferentes aplicaciones en el campo de las finanzas y en

la tercera (caps. 11 y 12) se describen varias aplicaciones en mercadotecnia (marketing).

4.2.1. Impago de hipotecas

Para trabajar con Carmen en este ámbito usamos una base de datos de Caja Madrid que contiene 2851 registros con información sobre operaciones de préstamo e hipotecas concedidas entre Diciembre de 2007 y Febrero de 2008. En particular, nos interesa conocer la recomendación dada por la Sociedad de Cobros sobre qué es más conveniente hacer en cada uno de los casos de impago.

Análisis de la base de datos

El trabajo previo al aprendizaje realizado con la base de datos es similar al explicado en la sección 4.1.1 con la salvedad de que, en este caso, se trabajó sólo con una parte de los registros contenidos en la base de datos. Los datos recogidos en la base de datos se refieren a tres tipos de operaciones (indicadas por el atributo 'marca'):

1. Esta operación y el resto de operaciones del cliente han sido cobradas.
2. La operación de préstamo se ha regularizado pero no el resto de operaciones del cliente.
3. Esta operación y el resto de operaciones del cliente pasan a otro ámbito de gestión, al cumplir 90 días del imcumplimiento de la operación.

Dado que nuestro objetivo se ceñía a los casos de morosidad, optamos por trabajar tan sólo con los casos de tipo 3, con lo que la base de datos original quedó reducida a 528 casos. Las variables seleccionadas para el aprendizaje son las siguientes[5]:

- *Canal*: canal a través del cual se ha formalizado la hipoteca.
- *Esf_orig*: porcentaje que supone la hipoteca sobre los ingresos del cliente. (Se discretizó en 4 intervalos de igual frecuencia).
- *Rama de actividad*: profesión del cliente.
- *Antigüedad*: antigüedad como cliente. (Se discretizó en 3 intervalos de igual anchura).
- *Cliente*: se considera cliente si tiene al menos 12 meses de antigüedad.
- *Localizado*: indica si ha sido posible localizar al cliente.
- *Mot_Iloc*: motivo de la ilocalización.
- *Agrupación Impago.*
- *Temporalidad*: temporalidad del impago, manifestada por el cliente.
- *Recomendación_SSCC*: Recomendación dada por la Sociedad de Cobros a Caja Madrid.

[5]Se muestra entre paréntesis la opción de discretización elegida en caso de que la variable sea numérica.

Resultados obtenidos

Con las opciones indicadas en el apartado anterior, la red obtenida tras el proceso de aprendizaje se muestra en la figura 4.6. Esta red nos permite ver las correlaciones existentes entre las distintas variables involucradas en el proceso de aprendizaje. Como en los casos anteriores, se ha usado Elvira para obtener más información acerca de las correlaciones encontradas. La red abierta en Elvira presenta el aspecto mostrado en la figura 4.7.

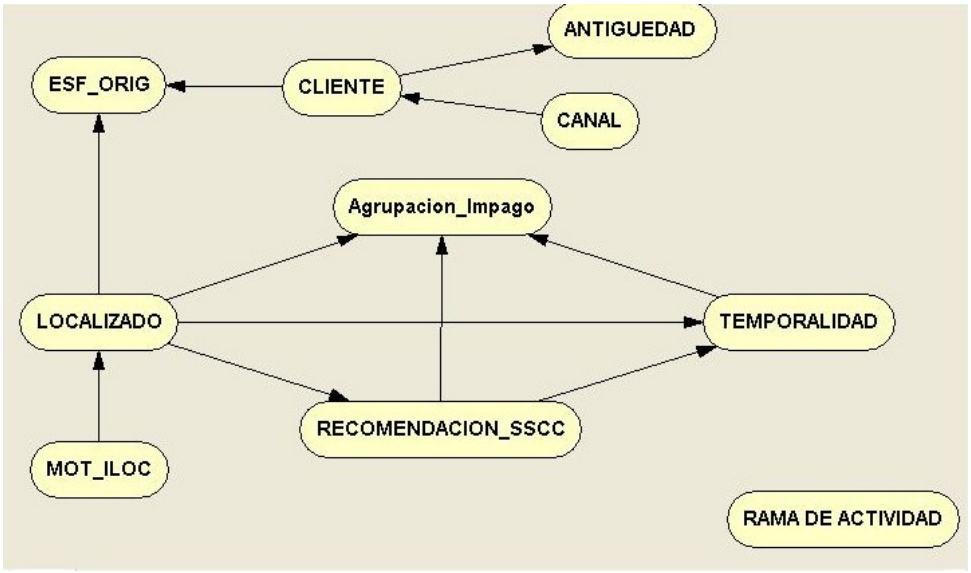


Figura 4.6: Red aprendida a partir de la base de datos de morosidad de hipotecas.

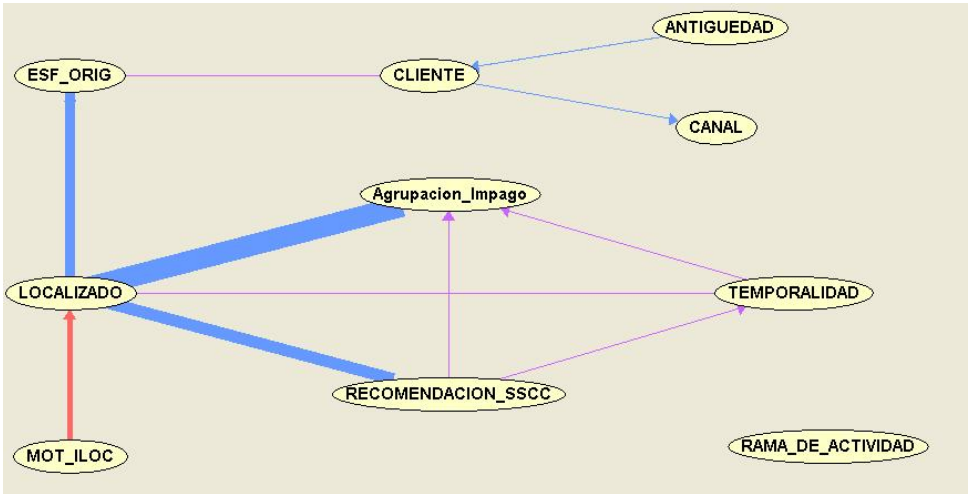


Figura 4.7: Red de morosidad de hipotecas, abierta en Elvira.

En este caso podemos observar, como en casos anteriores, algunas correlaciones obvias: Los atributos 'antigüedad' y 'cliente' están relacionados, lo cual era de esperar puesto que

se considera cliente a aquellas personas que tienen más de 12 meses de antigüedad. De este modo, cabría esperar que la correlación fuese muy fuerte, debido a que la relación es muy obvia, sin embargo, como comentamos, la variable antigüedad se discretizó en 3 intervalos de igual anchura, lo cual da lugar a un primer intervalo de 0 a 74, con lo que la correlación se suaviza enormemente puesto que los valores entre 12 y 74 (asociados al valor cliente) se toman como el mismo caso que los valores entre 0 y 12 (asociados a no ser cliente). Este hecho evidencia que la discretización de la variable 'antigüedad' no ha sido la más adecuada (la más adecuada habría sido la discretización en dos intervalos: $[0, 12]$ y $(12, +\infty)$). Sin embargo se ha preferido mantenerla en este ejemplo para mostrar los efectos que puede tener una mala discretización: donde había una correlación muy evidente, hemos obtenido una correlación muy débil. Además, dada la relación entre las variables 'cliente' y 'antigüedad', la primera de ellas es irrelevante, puesto que la información que contiene está contenida en la variable 'antigüedad', de modo que podría ser eliminada.

Otra de las correlaciones obvias, habiendo analizado el contenido de la base de datos, es la que se da entre las variables 'localizado' y 'mot_iloc'. Sin embargo, esta correlación apenas aporta información útil puesto que la variable 'mot_iloc' toma el valor 'Vacío' siempre que la variable 'localizado' toma el valor 'sí'. Como en el caso anterior, este hecho muestra un mal análisis previo de la base de datos. Es fácil ver que la variable 'localizado' no aporta ningún tipo de información (y sí aporta un incremento en el tiempo de aprendizaje), basta con tener el atributo 'mot_iloc' puesto que, en todos los casos, el estado 'vacío' se corresponde con el estado 'sí' de la variable 'localizado'. Como en el caso anterior, la variable 'localizado' se ha mantenido con el fin de ilustrar la importancia de realizar un correcto análisis previo de la base de datos.

Por otra parte, llama la atención la fuerte correlación entre las variables 'localizado' y 'recomendación_sscc'. A la vista de una correlación tan fuerte, parece que la variable 'localizado' podría bastar para predecir la recomendación adecuada. Sin embargo un análisis de los estados de cada variable nos hace desechar esa idea. La variable 'recomendación_sscc' tiene cuatro posibles valores, dos de los cuales ('quita' y 'dación') aparecen tan sólo 1 y 2 veces respectivamente en la base de datos, lo cual hace que aparezca una correlación mucho más fuerte de lo que en realidad es, ya que los valores bajos de 'recomendación_sscc' (que son, justamente, 'quita' y 'dación') se corresponden con valores altos de la variable 'localizado' (en esos 3 casos siempre toma el valor 'S'). De este modo podemos comprobar otra de las circunstancias que pueden sesgar el aprendizaje: la presencia de unos valores de baja frecuencia en la base de datos puede hacer aparecer correlaciones donde no las hay o, si las hay, fortalecerlas o debilitarlas. Por tanto, es muy recomendable analizar cuidadosamente los distintos valores de cada variable antes de comenzar el proceso de aprendizaje.

En este ejemplo también podemos ver correlaciones que resultan bastante obvias a la vista de los estados de las distintas variables: la correlación entre las variables 'localizado' y 'agrupación_impago' era de esperar, ya que la segunda tiene una serie de estados ('sin motivo', 'fuera de España' o 'no manifiesta') obviamente muy asociados al hecho de no haberse localizado al cliente.

En conclusión, este ejemplo ha servido no sólo para mostrar las posibles correlaciones existentes entre las variables del problema sino también para confirmar la importancia de un buen análisis previo de la base de datos con la que se trabaja. Ha quedado patente la necesidad de realizar una correcta discretización de las variables numéricas, una correcta selección de las variables que participan en el aprendizaje y un correcto preprocesado manual que permita detectar casos que pueden sesgar los resultados.

A continuación, en la figura 4.8, se muestra el resultado del aprendizaje una vez corregidos todos los errores comentados, para poder ver así las diferencias existentes.

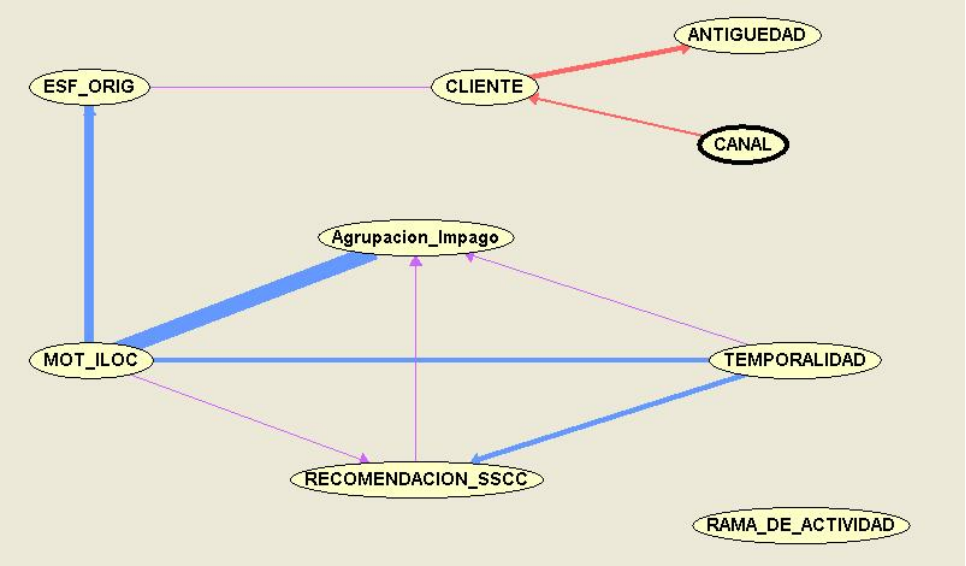


Figura 4.8: Red de morosidad de hipotecas obtenida tras la corrección.

La red obtenida confirma todos los comentarios realizados anteriormente. En primer lugar, como era de esperar, al eliminar la variable 'localizado', la estructura de la red es la misma, con la variable 'mot_iloc' ocupando su lugar (se mantienen las correlaciones (y su sentido e intensidad) con las varaibles 'esf_orig' y 'agrupación_impago' y el sentido y fuerza de la correlación con la variable 'recomendación_sscc' desaparecen por otras causas que comentamos más adelante). Además, la discretización aplicada a la variable 'antigüedad' (se decidio usar dos intervalos de igual frecuencia tras un breve análisis de los valores de la variable 'antigüedad' en la base de datos) se ajusta mucho mejor a los datos ya que, como podemos ver, la correlación obtenida ahora entre 'cliente' y 'antigüedad' es mucho más fuerte. Sin embargo, sigue sin ser todo lo fuerte que cabría esperar puesto que, como hemos señalado antes, la mejor discretización sería la dada por los intervalos: $[0, 12]$ y $(12, +\infty)$, pero para conseguir exactamente esa discretización, sería necesario usar la discretización manual, que aún no es accesible desde la interfaz gráfica.

Por otra parte, como suponíamos, la fuerte correlación existente entre las variables 'localizado' y 'recomendación_sscc' desaparece al eliminar los valores 'quita' y 'dación' de

esa segunda variable. Además, la eliminación de esos dos estados hace que aparezca una correlación con la variable 'temporalidad' que antes permanecía oculta. Esto no quiere decir que haya que eliminar de una base de datos aquellos registros que contengan valores que aparecen en contadas ocasiones, simplemente viene a decir que es conveniente tener en cuenta dichos casos a la hora de interpretar las correlaciones, mostrando, una vez más, la importancia de un cuidadoso análisis previo de la base de datos.

Capítulo 5

Conclusiones

Lo improbable es siempre si no de malo, como mínimo, de dudoso gusto.

Oscar Wilde, "La importancia de llamarse Ernesto".

Como ya sabíamos de antemano, la sinergia entre modelos gráficos y la teoría de la probabilidad constituye un núcleo fiable alrededor del cual construir sistemas expertos en ámbitos, como el de la medicina, en los que el elevado número de fuentes de incertidumbre hace necesario un tratamiento sitemático, fundamentado en una teoría matemática que indica cómo obtener las probabilidades implicadas y cómo combinarlas. En particular, los algoritmos de aprendizaje son una herramienta útil y eficaz para construir modelos a partir de bases de datos sin la ayuda de un experto. En cuanto al desarrollo e implementación de éstas técnicas, el presente trabajo deja patente la gran importancia y utilidad del seguimiento riguroso de una metodología basada en los fundamentos de la ingeniería del software para obtener un producto final de calidad.

En el campo teórico, la realización de este trabajo nos ha permitido conocer las principales ventajas y limitaciones de los MGP's. Sus principales ventajas son:

1. La mayor ventaja de los MGP frente a métodos alternativos para el tratamiento de la incertidumbre es su fundamento normativo, es decir, están basados en una teoría matemática consolidada que indica cómo obtener las probabilidades implicadas y trabajar con ellas. Al utilizar otros métodos, la única justificación existente es construir un modelo y comprobar que funciona.

2. Otra gran ventaja de estos modelos es que prácticamente todos los MGP desarrollados emplean razonamiento causal, lo cual permite tres tipos de razonamiento: abductivo (de los efectos a las causas), deductivo-predictivo (de las causas a los efectos) e intercausal (entre dos causas de un mismo efecto). Dicho de otro modo: se obtienen todas y únicamente las inferencias que están justificadas.

3. Esta capacidad de los MGP es consecuencia directa del tratamiento explícito de las dependencias e independencias condicionales. Se ha demostrado que los métodos

basados en reglas contenían hipótesis de independencia condicional más estrictas y más difíciles de justificar que las contenidas en los MGP.

4. Finalmente, frente a los llamados métodos de 'caja negra' como la regresión logística o las redes neuronales, los MGPs presentan una gran legibilidad, lo cual facilita su comprensión y análisis.

Por su parte, las mayores desventajas son:

1. Una de las principales desventajas es que muchos de los algoritmos existentes tienen complejidad exponencial para redes generales. Sin embargo, algunos de los algoritmos actuales son capaces de computar modelos bastante complejos en intervalos de tiempo razonables, e incluso existen aproximaciones y modelos simplificados que permiten abordar problemas de mayor tamaño.

2. Otro de los problemas es la dificultad para construir las redes, debida tanto a la escasez de bases de datos completas (o, al menos, lo suficientemente pobladas) como a las carencias de conocimiento que dificultan la construcción con ayuda de expertos humanos. Sin embargo, conviene señalar que ésta no es una deficiencia de los MGP, sino una dificultad intrínseca de los problemas que estamos abordando.

3. Por último, los MGPs presentan dos desventajas importantes frente a otro tipo de métodos como los basados en reglas: éstos últimos son capaces de controlar el razonamiento, fijando objetivos y generando las preguntas oportunas y son capaces de explicar el razonamiento, mientras que los MGPs carecen de estas capacidades.

5.1. Consecución de objetivos

Como señalamos en la sección 1.2, el principal objetivo de este trabajo de fin de máster es el desarrollo de un módulo de aprendizaje con la capacidad de ser utilizado para la implementación de nuevas aplicaciones y de ser fácilmente ampliado para poder probar nuevos algoritmos y métricas. El seguimiento de los principios básicos de la ingeniería del software ha desembocado en un módulo que cumple dichas características. El cuidadoso diseño explicado en el capítulo 3, en conjunción con la amplia documentación generada (tanto externa, como el capítulo de diseño de este mismo trabajo, como interna, en forma de comentarios explicativos en el código y páginas HTML generadas por la herramienta Javadoc de Sun[1]) facilita enormemente la tarea de implemetanción de nuevas técnicas. Todo este esfuerzo de diseño e implementación de los algoritmos de aprendizaje viene acompañado de una interfaz gráfica que permite manejar los distintos parámetros y opciones de una manera amigable e intuitiva, cosa que contribuye enormemente, junto a la gran extensibilidad del módulo, al objetivo de implantación de Carmen entre la comunidad científica.

En este sentido, la principal conclusión que podemos extraer es que el uso de las técnicas de la ingeniería del software es altamente recomendable para dar lugar a productos de calidad que sean fácilmente reusables, mantenibles y ampliables. Otros intentos llevados

[1] `http://java.sun.com/j2se/javadoc/`

a cabo en los últimos años para desarrollar herramientas de manejo de modelos gráficos probabilistas han fracasado en su intento de imponerse, quizás debido a la dificultad de mantenimiento y falta de fiabilidad provocadas por un diseño y desarrollo alejado de los principios básicos de la ingeniería del software. Ésta es la principal aportación de Carmen, de ahí que se espera que tenga un mayor éxito de implantación y pueda ser de gran ayuda a la comunidad científica.

En cuanto a los objetivos secundarios, el presente trabajo recoge, como se pretendía, de forma breve el estado de la técnica en aprendizaje de redes bayesianas, que puede servir como una introducción general al tema, permitiendo al lector interesado profundizar en los distintos temas tratados, a través de las referencias indicadas a lo largo del mismo.

Además, la aplicación del módulo de aprendizaje a casos reales ha permitido comprobar la utilidad del módulo desarrollado a la hora de encontrar relaciones de dependencia e independencia entre las distintas variables (como señalamos en la sección 4.1, una de las principales ventajas de las redes bayesianas es que pueden representar correctamente las relaciones de dependencia e independencia), permitiendo así encontrar nuevas relaciones o descartar algunas hipótesis sobre las mismas. Del mismo modo, se han comprobado en la práctica algunas de las ventajas y desventajas de los MGP's comentadas anteriormente. En resumen, los distintos experimentos llevados a cabo en el capítulo 4 nos han permitido comprobar varias de las utilidades del módulo desarrollado:

- Detectar correlaciones entre las distintas variables.
- Detectar relaciones de independencia entre las distintas variables.
- Confirmar o rechazar hipótesis sobre las relaciones presentes entre las variables.
- Explicar ciertas correlaciones que podían parecer inexplicables, gracias a la detección de nuevas varaibles que no habían sido tenidas en cuenta.
- Por último, y aunque no haya sido comentado anteriormente porque queda fuera del módulo de aprendizaje, las redes obtenidas pueden ser utilizadas para aplicar distintos mecanismos de inferencia y predecir el valor de algunas variables a partir de los valores observados en otras.

5.2. Valoración del módulo de aprendizaje

Frente a otro tipo de técnicas de aprendizaje automático, las cualidades y limitaciones del módulo de aprendizaje desarrollado son las propias de los modelos gráficos probabilistas y de sus técnicas de aprendizaje, de modo que a continuación recogeremos, a modo de resumen, las principales ventajas y desventajas de éste módulo de aprendizaje frente a otros sistemas similares.

5.2.1. Logros

Como venimos señalando, las principales cualidades de este módulo son las que le aporta el hecho de haber sido diseñado e implementado siguiendo una metodología basada

en los principios básicos de la ingeniería del software. Mientras otros sistemas similares carecen de varias de ellas, la herramienta Carmen en general y el módulo de aprendizaje en particular presentan las siguientes características:

- Robustez y fiabilidad: El detallado seguimiento de la metodología comentada en la sección 1.3 garantiza estas cualidades no sólo para la parte implementada actualmente sino también para futuras ampliaciones.
- Extensibilidad: Esta es una de las principales ventajas frente a otras herramientas similares. La facilidad de ampliación de Carmen es mucho mayor debido al seguimiento riguroso de las técnicas de ingeniería del software y a un especial cuidado de la documentación tanto interna como externa.
- Usabilidad: La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso. En este sentido se han realizado esfuerzos tanto en el desarrollo de una interfaz gráfica intuitiva y fácil de usar, como en el desarrollo de documentación que permita comprender y reusar fácilmente el código fuente.

Por otra parte, el cuidado desarrollo de la interfaz de usuario atendiendo a las características citadas en la sección 3.5 supone también un punto importante para la consecución de una buena aceptación de la herramienta Carmen.

5.2.2. Limitaciones

En este momento, una de las principales limitaciones del aprendizaje en Carmen es su lentitud. A día de hoy no se ha realizado un estudio en profundidad comparando con otras herramientas similares, simplemente se ha llevado a cabo una comparación superficial con los programas Elvira y Weka, resultando ser Carmen el más lento (aunque la compración con Elvira no es del todo fiable puesto que no permite ejecutar el mismo algoritmo de aprendizaje implementado en Carmen). Este estudio comparativo superficial nos lleva a pensar que sería conveniente en un futuro dedicar cierto esfuerzo a mejorar la eficiencia de Carmen mediante el uso de una caché que permita reducir el número de cálculos realizados al medir la calidad de las sucesivas redes.

Además, en la versión actual del sistema, la escalabilidad es todavía un problema importante. Se ha comprobado que, a pesar de utilizar estructuras muy eficientes en términos de consumo de memoria (como la explicada en la sección 3.4), el proceso de aprendizaje requiere una gran cantidad de memoria al trabajar con un gran número de variables y de estados por cada variable. En algunos de los casos probados, el consumo de memoria es tan elevado que hace imposible el aprendizaje. Una primera solución implementada consiste en dotar al módulo de la capacidad de discretizar variables numéricas, de modo que el número de estados de dichas variables se reduce enormemente, pero aún así queda pendiente el tratamiento de redes con un número muy elevado de variables.

Otra de las limitaciones actuales del sistema es que no es capaz de mostrar el sentido ni la intensidad de las relaciones representadas por la red. En otras herramientas como Elvira se muestra el sentido de la correlación y la intensidad de la misma mediante un

código de colores de los enlaces de la red, como hemos podido comprobar en el capítulo 4. Esta es una característica muy deseable puesto que aporta información que puede resultar muy relevante para el análisis de las redes obtenidas.

5.3. Trabajo futuro

El presente Trabajo de Fin de Máster deja abiertas varias líneas de investigación que podemos dividir en dos tipos: las que se refieren a la parte teórica y las referidas a la mejora del módulo implementado.

Teoría

Las líneas de trabajo en un ámbito teórico son muy amplias y van desde el desarrollo de nuevos algoritmos de aprendizaje más eficientes hasta el tratamiento teórico del problema de la escalabilidad, pasando por otros temas que en este trabajo se han señalado brevemente como el tratamiento de valores ausentes, el aprendizaje de redes causales, la presencia de variables ocultas o el desarrollo de clasificadores basados en redes bayesianas.

Por otra parte, sería interesante realizar un estudio del funcionamiento de los algoritmos de aprendizaje en otros ámbitos además de los ya estudiados, analizando las ventajas e inconvenientes que presentan las redes bayesianas en cada uno de los ámbitos, ampliando así el conocimiento general que se tiene de ellas.

Implementación

En cuanto al aspecto puramente de implementación, los primeros objetivos a abordar han de ir encaminados a superar las limitaciones mencionadas en el apartado anterior. En primer lugar se ha de mejorar la eficiencia del sistema. Aunque la versión actual permite trabajar con redes con un número de variables y estados no muy elevado en un tiempo razonable, en un primer análisis superficial se ha comprobado que el proceso de aprendizaje es más lento que en otras herramientas similares. En este sentido se pretende añadir una memoria caché al módulo de aprendizaje que permita reducir los cálculos realizados, mejorando así su eficiencia.

Relacionado con lo comentado en el párrafo anterior aparece el problema de la escalabilidad. Al trabajar con redes de un gran número de variables y de estados por cada variable, el consumo de memoria se dispara. La discretización de variables numéricas permite reducir el número de estados de dichas variables pero queda pendiente la implementación de soluciones para el trabajo con un elevado número de variables.

Además, en relación con otra de las limitaciones comentadas, sería interesante dotar a Carmen de la capacidad de mostrar el sentido y la intensidad de las relaciones representadas por la red. Incluso podría mejorarse dicha funcionalidad permitiendo que se ordenen los estados de las variables numéricas de menor a mayor haciendo así más intuitiva la

interpretación de los colores de los enlaces (evitando el problema que supone tener en cuenta la ordenación de los estados, comentado en la página 56).

Otra de las líneas de trabajo obvias, en la que hay una cantidad de trabajo prácticamente ilimitado, consiste en la ampliación del módulo de aprendizaje mediante el desarrollo de nuevos algoritmos y métricas. Sin embargo, esta línea de trabajo no es tan prioritaria como las anteriores, ya que la gran facilidad de extensión de la que está dotada el módulo de aprendizaje permite a cualquier persona desarrollar fácilmente los algoritmos y métricas en los que esté interesado, de modo que no es demasiado importante implementar un gran abanico de estos elementos.

Además, podría resultar interesante dotar al módulo de aprendizaje de un sistema de evaluación que permita comparar la calidad de distintas redes bayesianas al clasificar un nuevo conjunto de datos. En la versión actual, resulta muy difícil comprobar la calidad de los modelos generados, cosa que puede ser muy útil a la hora de comparar los resultados de distintos algoritmos de aprendizaje en distintos ámbitos.

Apéndice A

Manual de Usuario

A.1. Introducción

El módulo de aprendizaje de Carmen es una herramienta de aprendizaje automático de redes bayesianas desarrollada en la UNED, y enmarcada dentro del software para el manejo de redes bayesianas Carmen. La funcionalidad principal que ofrece este módulo es la del aprendizaje de redes bayesianas a partir de bases de datos en distintos formatos (.dbc, .arff o .xls), ofreciendo además otras funcionalidades complementarias que permiten el manejo y preprocesamiento de los datos. Todas estas opciones se ofrecen al usuario a través de una interfaz amigable e intuitiva que facilita el manejo de la herramienta.

Tanto éste módulo de aprendizaje como la herramienta Carmen en su conjunto están desarrollados íntegramente en Java, de modo que el único requerimiento es tener instalado el "Java Runtime Environment", disponible en: `http://www.java.com/es/download/index.jsp`.

Para acceder a la interfaz del módulo de aprendizaje de Carmen basta con seleccionar la opción "Aprendizaje" dentro del menú desplegable "Aprendizaje", como se muestra en la figura A.1, o pulsar las teclas Ctrl + N.

Al acceder al módulo de aprendizaje, se nos muestra una pantalla inicial en la que podemos distinguir dos pestañas: La pestaña 'General'(figura A.2) permite cargar y guardar ficheros de bases de datos, así como cargar redes previamente creadas para utilizarlas como modelo durante el aprendizaje. La pestaña 'Variables'(figura A.3) permite seleccionar las variables que van a formar parte de la red y elegir las distintas opciones de preprocesado para cada una de ellas. Además, en la parte inferior de la ventana, se da la opción de elegir el algoritmo de aprendizaje, la métrica y el valor del parámetro alfa a usar en la etapa de aprendizaje paramétrico.

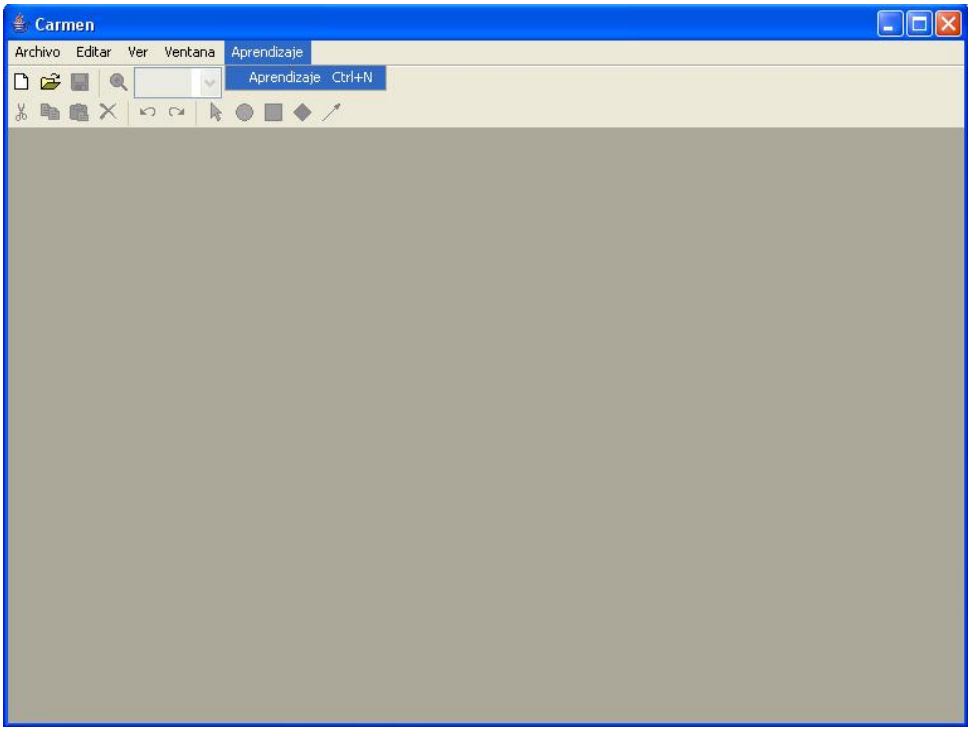


Figura A.1: Lanzamiento del módulo de aprendizaje de Carmen.

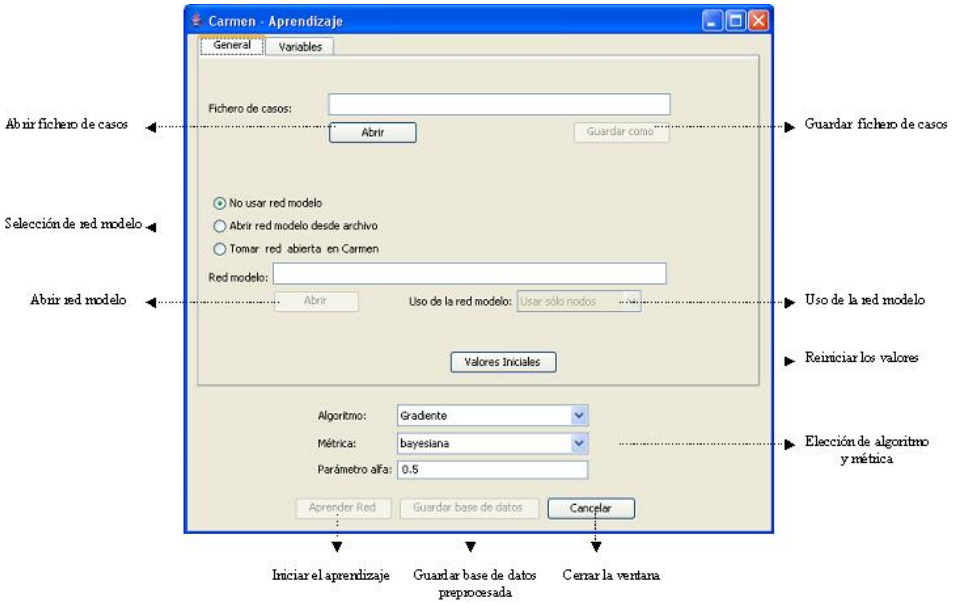


Figura A.2: Aspecto inicial de la pestaña 'General'.

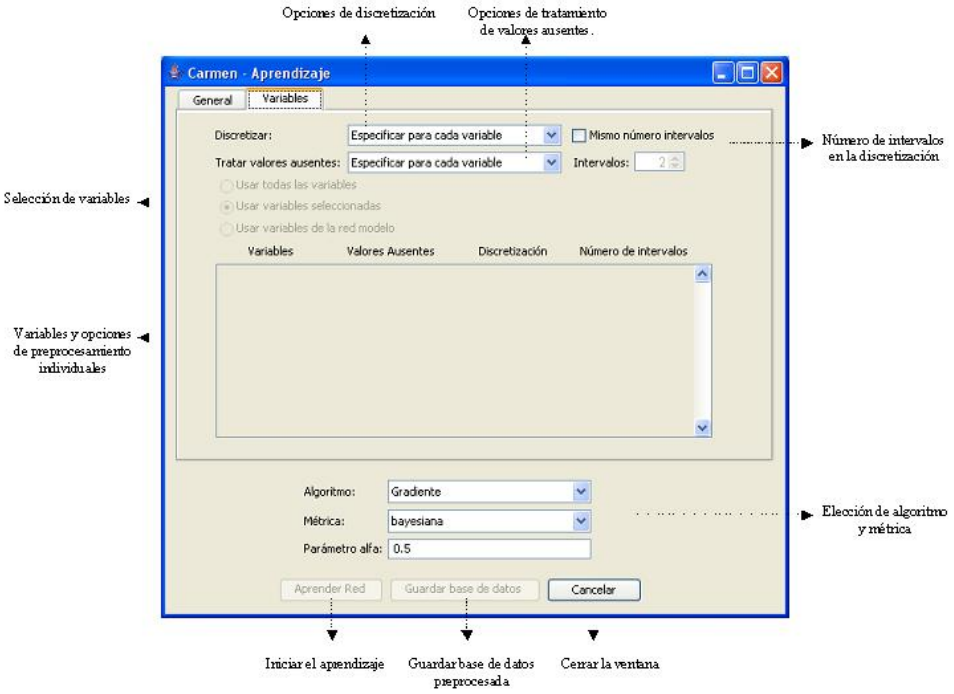


Figura A.3: Aspecto inicial de la pestaña 'Variables'.

A.2. Cargar/Guardar bases de datos

En la parte superior de la pestaña 'General' se nos ofrece la posibilidad de cargar una base de datos. Para ello basta con pinchar en el botón 'Abrir' y elegir el fichero correspondiente en el menú que se despliega. Los formatos de bases de datos aceptados por Carmen son[1]:

- Formato .dbc: Formato de bases de datos usado en Elvira.
- Formato .arff: Formato de bases de datos usado en Weka.
- Formato .xls: Formato usado en Microsoft Excel.

Una vez elegido el fichero a partir del cual se quiere cargar la base de datos, la opción 'Guardar como' está disponible para guardar la base de datos previamente cargada, en uno de los formatos indicados. Esta función puede ser útil, entre otras cosas, para transformar fácilmente bases de datos entre distintos formatos y así, poder comparar los modelos obtenidos en Carmen, Elvira y Weka.

A.3. Usar red modelo

El uso de una red modelo permite al usuario definir una red que puede ser utilizada simplemente para colocar en pantalla los nodos de la red obtenida durante el aprendizaje,

[1]Para una descripción más detallada de los distintos tipos de formatos aceptados véase la sección A.8 de este mismo manual.

para iniciar el proceso de aprendizaje a partir de esa red o para iniciar el aprendizaje a partir de esa red manteniendo los enlaces iniciales. En cualquier caso, el uso de una red modelo es totalmente opcional, y el aprendizaje puede realizarse sin necesidad de usarla, con el único inconveniente de que los nodos de la red resultante apareceran apilados en la esquina superior izquierda de la pantalla donde se muestra la red, siendo necesario recolocar los nodos en pantalla manualmente.

En la parte media de la pestaña 'General' se dan tres opciones para seleccionar una red modelo:

- 'No usar red modelo': No se usará ninguna red modelo durante el aprendizaje.
- 'Abrir red modelo desde archivo': Tras elegir esta opción, es necesario cargar un fichero con una descripción de la red modelo en formato '.elv' o '.xml'[2]. Para ello, basta con pinchar en el botón Abrir que se encuentra debajo de las opciones de selección de red modelo, y seleccionar el archivo correspondiente.
- 'Tomar red abierta en Carmen': Al elegir esta opción se usa como red modelo la red que esté abierta en ese momento en la pantalla principal de Carmen. En caso de que no haya ninguna red abierta se devuelve un mensaje de error al usuario y se marca la opción 'No usar red modelo'.

Una vez seleccionada una red modelo, existen tres maneras distintas de utilizar dicha red. Estas opciones se pueden seleccionar en el 'ComboBox' que aparece bajo el campo de texto correspondiente a la red modelo. Las tres opciones posibles son:

- 'Usar sólo nodos': Con esta opción seleccionada, la red modelo sólo se utiliza una vez finalizado el aprendizaje, para obtener las posiciones de los nodos en pantalla.
- 'Usar enlaces iniciales': Esta opción permite, además de obtener las posiciones en pantalla de los nodos de la red, iniciar el proceso de aprendizaje a partir de la red seleccionada en lugar de usar una red sin enlaces.
- 'Fijar enlaces iniciales': Esta opción tiene los efectos de las dos opciones anteriores, con el añadido de que los enlaces de la red modelo no pueden ser eliminados durante la etapa de aprendizaje (mientras que en el caso anterior, esto sí podía suceder).

Es importante señalar, que para utilizar la información de los nodos de la red modelo en la red aprendida se toma como referencia el nombre de cada variable, es decir, la información de la varaible 'A' de la red modelo pasará a la variable 'A' de la red aprendida, de modo que es relevante prestar atención a los nombres de las variables en ambas redes.

A.4. Selección de variables

Una vez que se ha elegido una base de datos en la pestaña 'General', la pestaña 'Variables' presenta un aspecto similar al de la figura A.4. Al haberse cargado una base de

[2]Para una descripción más detallada de los distintos tipos de formatos aceptados véase la sección A.8 de este mismo manual.

datos, se muestran en la parte central de la pestaña las variables presentes en dicha base de datos seguidas de tres elementos que permiten seleccionar individualmente la opción de tratamiento de valores ausentes (desactivada si dicha variable no tiene valores ausentes en la base de datos), la opción de discretización (desactivada si la variable no es numérica) y el número de intervalos a usar en la discretización. Además, se han activado las opciones de selección de variables y de aprendizaje.

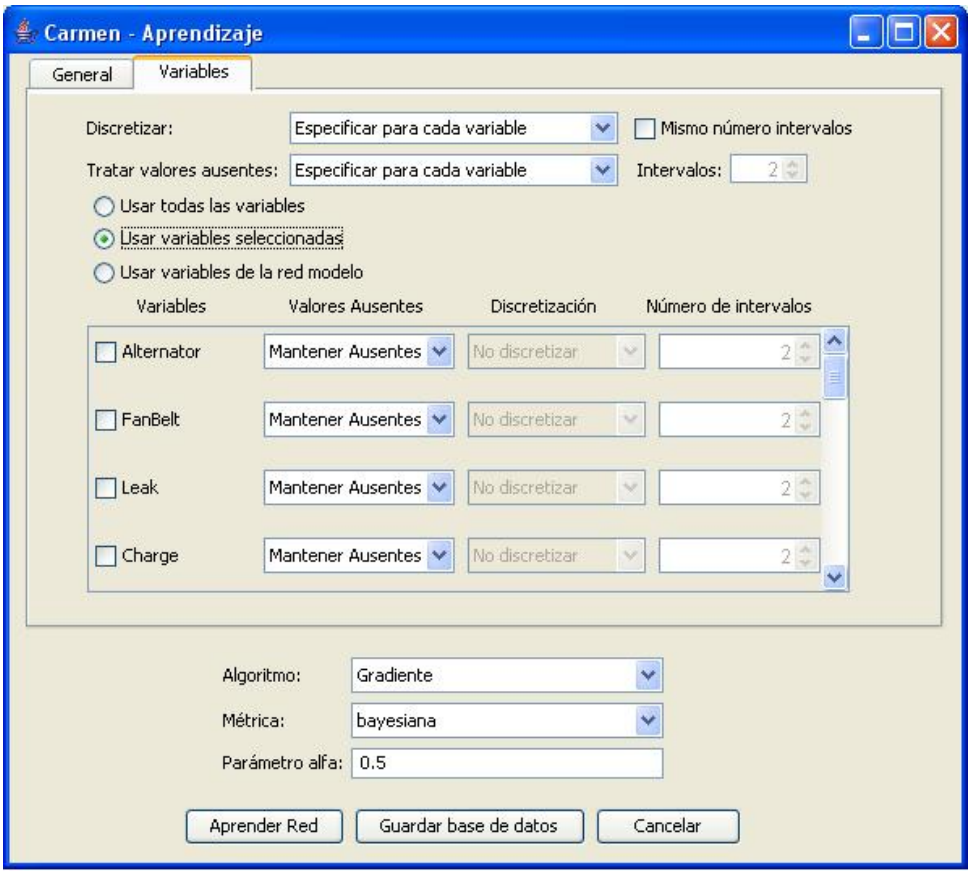


Figura A.4: Aspecto de la pestaña 'Variables' una vez cargado un fichero de casos.

Para elegir las variables que queremos usar en el aprendizaje tenemos tres opciones, que podemos elegir en la parte superior izquierda de la ventana:

- 'Usar todas las variables': Esta opción selecciona automáticamente todas las variables de la base de datos.
- 'Usar variables seleccionadas': Esta opción permite seleccionar una a una las variables que se desean usar en el aprendizaje. Para seleccionar una de las variables basta marcar el 'Checkbox' que aparece a la izquierda del nombre de la variable.
- 'Usar variables de la red modelo': Esta opción utiliza en el aprendizaje, las variables que tienen el mismo nombre que las que aparecen en la red modelo. Esta opción

está disponible únicamente si se ha seleccionado utilizar una red modelo en la pestaña 'General'[3].

Es importante señalar que, antes de empezar el proceso de aprendizaje, es necesario haber seleccionado las variables que se quiere que intervengan en ese proceso. En caso contrario, el aprendizaje no puede llevarse a cabo.

A.5. Preprocesamiento

El preprocesamiento de las variables a utilizar es un paso previo importante antes de iniciar el proceso de aprendizaje. Tanto la eficacia como la eficiencia de dicho proceso pueden verse afectadas por el uso del preprocesamiento. En Carmen, el preprocesamiento de las variables se divide en tratamiento de los valores ausentes y discretización de variables continuas. Cada una de estas dos ramas presentan distintas opciones que se comentan en las siguientes secciones.

En la parte superior de la pestaña 'Variables' se permite elegir una misma opción de tratamiento de valores ausentes y de discretización para todas las variables. En caso de que se quieran elegir distintas opciones para cada una de las variables, se ha de hacer uso de los 'ComboBox' situados a la derecha de los nombres de cada una de las variables.

A.5.1. Tratamiento de valores ausentes

Existen dos opciones para el tratamiento de valores ausentes:

- 'Eliminar registros': Se eliminan de la base de datos aquellos registros que contengan un valor ausente en alguna de las variables para las que se ha elegido esta opción. Es importante tener en cuenta que la elección de esta opción puede tener consecuencias muy negativas en el aprendizaje dependiendo de la densidad de la base de datos. Si la base de datos contiene gran cantidad de valores ausentes, la elección de esta opción puede reducir el número de casos para el aprendizaje hasta valores demasiado pequeños, reduciéndose así enormemente la calidad del aprendizaje. Por tanto, antes de elegir esta opción, es recomendable tener cierto conocimiento de la base de datos para evitar sesgar el aprendizaje de una manera muy importante.
- 'Mantener valores ausentes': Se mantienen los valores ausentes que aparecen en la base de datos, añadiendo a cada variable para la que se haya seleccionado esta opción, un nuevo estado que recoge dichos valores ausentes.

En la interfaz gráfica, en la parte superior de la pestaña 'Variables' se muestra un 'ComboBox' en el que se dan tres opciones (véase la figura A.5). La opción por defecto es 'Seleccionar para cada variable', la cual nos permite seleccionar la opción de tratamiento de valores ausentes para cada variable independientemente. Para ello, se ha de hacer uso del 'comboBox' situado a la derecha del nombre de cada variable. Además de dicha opción, se muestran otras dos opciones: 'Eliminar registros' y 'Mantener valores ausentes' (cuyo

[3]Véase la sección A.3 para obtener información sobre como cargar una red modelo.

efecto es el comentado anteriormente) que permiten aplicar la opción seleccionada a todas las variables de la base de datos. De este modo, si se quiere utilizar una misma política de tratamiento de valores ausentes para todas las variables de la base de datos es más cómodo seleccionar una de estas dos opciones, sin embargo, si se quieren mantener los valores ausentes para algunas variables y eliminarlos para otras, será necesario elegir la opción 'Seleccionar para cada variable' y a continuación, seleccionar la opción deseada para cada variable en el 'ComboBox' que aparece a la derecha de su nombre.

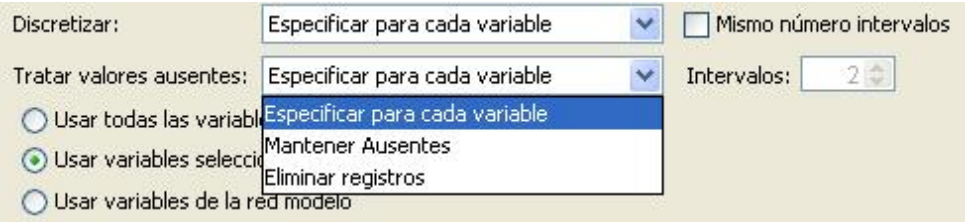


Figura A.5: Opciones de tratamiento de valores ausentes.

A.5.2. Discretización de variables continuas

La discretización de variables sólo tiene sentido cuando se está tratando con variables numéricas, de modo que esta opción queda desactivada para varaibles categóricas y para variables que tengan tan sólo dos estados numéricos, ya que se entiende que una variable con estados, por ejemplo, '0' y '1' no es una variable discretizable.

El módulo de aprendizaje de Carmen da cuatro opciones para la discretización de variables:

- 'No discretizar': Tanto los estados de la variable como los casos almacenados en la base de datos permanecen en su estado original.
- 'Según red modelo': Los nuevos estados de la variable se copian de la variable que tiene el mismo nombre en la red modelo. En caso de que dicha variable no exista, o su discretización no coincida con los datos de la base de datos (supóngase por ejemplo que la discretización de la variable en la red modelo está formada por los intervalos (0, 1) y [1, 2) y en nuestra base de datos aparece un valor de 17 para esa variable) se muestra un mensaje de error y se mantienen la variable original.
- 'Igual frecuencia': La variable se discretiza en intervalos de, aproximadamente, igual frecuencia. El hecho de que los intervalos tengan la misma frecuencia o no, depende de la distribución de los valores de la variable en la base de datos y del número de intervalos seleccionado. Supongamos que tenemos una variable con 8 estados que en la base de datos aparecen dos veces cada uno (distribución uniforme), si se nos pide realizar cuatro intervalos de igual frecuencia, los intervalos resultantes, efectivamente, tendrán la misma frecuencia. Sin embargo, si el primero de los valores apareciera en la base de datos 25 veces y los demás una sola vez, tendríamos un intervalo de frecuencia

25 y tres intervalos de frecuencia mucho menor. De ahí que resulte muy conveniente conocer la distribución de los datos antes de realizar este tipo de discretización.

- 'Igual anchura': La variable se discretiza en intervalos de la misma anchura.

La manera de seleccionar cada una de estas opciones en la interfaz gráfica es similar a la explicada para el tratamiento de valores ausentes. En la parte superior de la pestaña 'Variables' se muestra un 'ComboBox' en el que se dan cinco opciones (véase la figura A.6). La opción por defecto es 'Seleccionar para cada variable', la cual nos permite seleccionar la opción de discretización para cada variable independientemente. Para ello, se ha de hacer uso del 'ComboBox' situado a la derecha del nombre de cada variable. Además de dicha opción, se muestran las cuatro opciones comentadas anteriormente, que permiten aplicar la opción seleccionada a todas las variables de la base de datos. De este modo, si se quiere utilizar una misma política de discretización para todas las variables de la base de datos es más cómodo seleccionar una de estas cuatro opciones, en caso contrario será necesario elegir la opción 'Seleccionar para cada variable' y a continuación, seleccionar la opción deseada para cada variable en el 'ComboBox' que aparece a la derecha de su nombre.

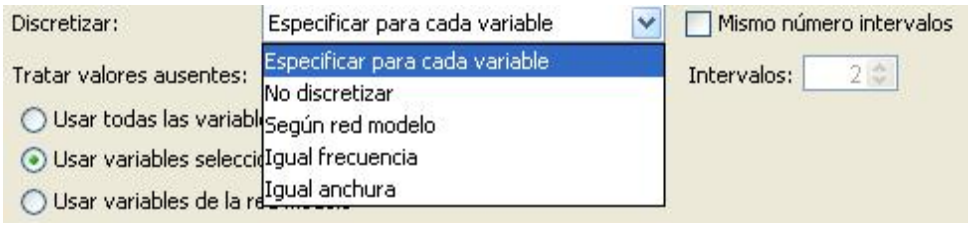


Figura A.6: Opciones de discretización.

Es importante señalar que para aquellas variables que se ha optado discretizar, no se permite mantener sus valores ausentes, pues no tiene mucho sentido tener en una variable discretizada un estado que no esté asociado a ningún intervalo. De este modo, al seleccionar una opción de discretización (obviamente, distinta de 'No discretizar'), se marcará la opción de 'Mantener valores ausentes' para el tratamiento de datos.

A.5.3. Guardar base de datos preprocesada

Una vez que se han elegido las distintas opciones de preprocesado, Carmen da la opción de guardar la base de datos preprocesada en alguno de los formatos comentados en la sección A.8. Para ello, basta con pinchar en el botón 'Guardar base de datos' que aparece en la parte inferior de la ventana y escribir el nombre del fichero donde se desea guardar la base de datos, en el cuadro de texto emergente.

A.6. Algoritmos y Métricas

En la parte inferior de la ventana del módulo de aprendizaje, común a las dos pestañas ya comentadas, se nos da la posibilidad de elegir el algoritmo de aprendizaje a utilizar, la métrica que se usará en dicho algoritmo y el parámetro alfa usado en la etapa de

aprendizaje paramétrico. En principio, cualquier combinación de algoritmos y métricas es posible, y se deja libertad al usuario para experimentar con los valores de cada uno de estos parámetros. En cuanto al parámetro alfa, es conveniente utilizar un valor real entre 0 y 1.

A.7. Aprendizaje

Una vez seleccionadas las variables que han de participar en el aprendizaje, sus opciones de preprocesamiento, el algoritmo a utilizar y la métrica, basta con pulsar el botón 'Aprender red' para que comience el proceso de aprendizaje. La duración de este proceso varía con el número de variables seleccionadas y el número de estados de cada variable, de ahí que, en muchos casos, resulta conveniente discretizar las variables para obtener mejores prestaciones tanto en términos de consumo de tiempo como en términos de consumo de memoria.

Una vez finalizado el aprendizaje, si no se ha producido ningún error, se nos muestra la pantalla principal de Carmen con la red aprendida. En caso de que se halla optado por utilizar una red modelo, los nodos cuyas variables se llamen igual que las variables de la red modelo, se colocaran en la posición que tenían en dicha red, mientras que los nodos que no aparecen en la red modelo quedan apilados en la esquina superior izquierda (para colocarlos en la pantalla basta con pinchar sobre ellos y arrastrar). En caso de que no se halla optado por utilizar una red modelo, todos los nodos apareceran sobrepuestos en la esquina superior izquierda de la pantalla, siendo necesario recolocarlos para visualizar correctamente la red obtenida (en la figura A.7 puede verse un ejemplo de red aprendidad con los nodos ya colocados en pantalla).

A.8. Formatos aceptados

El módulo de aprendizaje de Carmen puede trabajar con dos tipos de ficheros: ficheros de bases de datos, que almacenan los casos a partir de los cuales se realiza el aprendizaje, y fichero de redes modelo, a partir de los cuales se obtiene la información de las redes usadas como modelo en el proceso de aprendizaje[4]. Los formatos aceptados para cada uno de los tipos de fichero, así como las restricciones a cada uno de ellos se comentan a continuación.

A.8.1. Formatos de bases de datos

- **Formato '.xls'**: El formato de un fichero de casos '.xls' aceptado por Carmen es el siguiente:

  La primera línea ha de contener los nombres de las variables implicadas, una en cada celda sin dejar celdas en blanco entre ellas. En cada una de las líneas siguientes se ha de especificar un caso, incluyendo en cada celda el valor de una variable en el mismo orden en el que aparecen los nombres de las variables. Es decir, en cada columna ha de aparecer el nombre de la variable y todos los valores que toma dicha variable en los distintos casos registrados en el fichero. Cada uno de los valores de las variables

[4]Véase la sección A.3 para obtener más información sobre el uso de redes modelo.

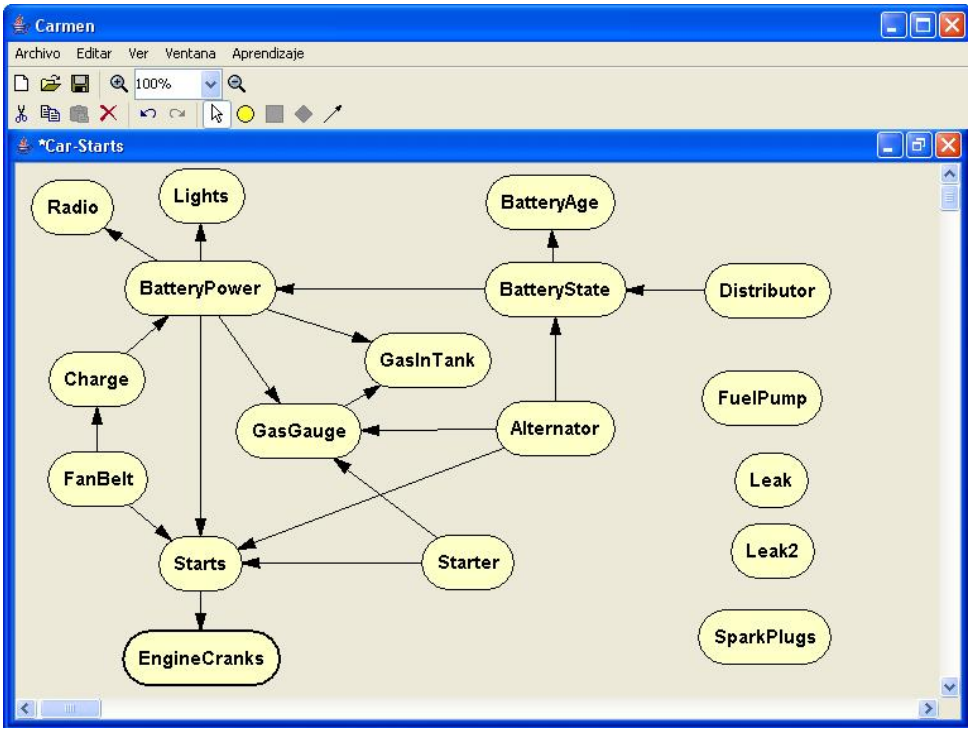


Figura A.7: Red resultante tras recolocar los nodos en pantalla.

puede especificarse como una cadena de caracteres o un entero. Para representar un valor ausente basta con dejar la celda correspondiente vacía.

- **Formato '.arff'**: Éste es el formato de bases de datos utilizado por Weka. Una descripción del formato '.arff' puede verse en `http://www.cs.waikato.ac.nz/~ml/weka/arff.html`.

  Las únicas restricciones que impone Carmen a dicho formato son:

  - No se admiten atributos de tipo 'numeric', 'string' ni 'date'.
  - No se admiten ficheros de casos dispersos ('Sparse ARFF files').

- **Formato '.dbc'**: Formato de bases de datos utilizado por Elvira. Las propiedades generales son semejantes a las definidas para las redes y diagramas (formato .elv explicado a continuación). Al comienzo del fichero se definirá el conjunto de nodos o variables de las que consta el problema, tipología, comentarios e información adicional que pueda resultar de interés. También deberá incluirse el número de instancias o casos que existen en el fichero, número que tendrá que coincidir con el número de líneas que más tarde se incluyan como información.

  Una vez definidas todas las variables se incluyen los casos que conforman el cuerpo de conocimiento de la base de datos. Cada instancia corresponderá con una línea, y, por cada línea, se incluirá el valor de esa instancia para cada una de las variables

definidas en el preámbulo del fichero. Los valores que tome cada variable pueden ser separados bien por comas, o bien dejando un espacio en blanco.

La única restricción que impone Carmen es que no se pueden leer variables continuas. Para introducir una variable continua se ha de definir como variable discreta y especificar uno a uno todos sus posibles estados.

A.8.2. Formatos de redes modelo

- **Formato '.elv'**: Formato para guardar redes utilizado en Elvira. Una descripción del formato '.elv' puede verse en `http://leo.ugr.es/elvira/devel/Formato/formato.html`
- **Formato '.xml'**: Se está desarrollando un formato propio para Carmen en '.xml' cuya descripción aún no está disponible. En futuras versiones de este manual se incluirá la descripción completa de este formato.

Bibliografía

[1] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[2] M. Arias and F. J. Díez. Carmen: An open source project for probabilistic graphical models. Hirtshals, Denmark. Pendiente de publicación.

[3] J. Bertrand. *Calcul des probabilités*. Gauthier-Villars et fils, París, 1889.

[4] R. Bouckaert. Bayesian networks in Weka. Technical Report 14/2004, Computer Science Department, University of Waikato, New Zealand, 2004.

[5] W. Buntine. Theory refinement on Bayesian networks. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence (UAI'91)*, pages 52–60, Los Angeles, CA, 1991. Morgan Kaufmann, San Mateo, CA.

[6] G. Casella and E. I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[7] E. Castillo, J. M. Gutiérrez, and A. S. Hadi. *Sistemas Expertos y Modelos de Redes Probabilísticas*. Academia de Ingeniería, Madrid, 1997.

[8] D. M. Chickering. Search operators for learning equivalence classes of Bayesian network structures. Technical Report, R231, UCLA Cognitive Systems Laboratory, Los Angeles, 1995.

[9] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–348, 1992.

[10] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[11] R. Descartes. *Discurso del método; estudio preliminar, traducción y notas de Bello Reguera*. ed. TECNOS, Madrid, 2003.

[12] F. J. Díez. *Sistema Experto Bayesiano para Ecocardiografía*. PhD thesis, Dpto. Informática y Automática, UNED, Madrid, 1994. In Spanish.

[13] F. J. Díez. Aplicaciones de los modelos gráficos probabilistas en medicina. In J. A. Gámez and J. M. Puerta, editors, *Sistemas Expertos Probabilísticos*, pages 239–263. Universidad de Castilla-La Mancha, Cuenca, 1998.

[14] F. J. Díez. Teoría probabilista de la decisión en medicina. Informe Técnico CISIAD-07-01, UNED, Madrid, 2007. In Spanish.

[15] B. Eckel. Thinking in patterns: Problemsolving techniques using java. Disponible en internet, 2003.

[16] P. Felgaer, P. Britos, J. Sicre, A. Servetto, R. García-Martínez, and G. Perichinsky. Optimización de redes bayesianas basada en técnicas de aprendizaje por instrucción. In *Proceedings del VIII Congreso Argentino de Ciencias de la Computación*, 2003.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, editors. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 2005.

[18] D. Geiger and D. Heckerman. A characterization of the Dirichlet distribution with application to learning Bayesian networks. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 196–207. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[19] A. E. Gelfand and A. F. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409, 1990.

[20] W. R. Gilks and P. Wild. Adaptive rejection sampling for the gibbs sampling. *Journal of the Royal Statistical Society, Series C*, 41:337–348, 1992.

[21] C. Glymour and G. F. Cooper. *Computation, causation and discovery*. The MIT Press, Cambridge, Massachusetts, 1999.

[22] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62, 1993.

[23] P. S. Laplace. *Théorie Analytique des Probabilités*. Courcier, París, 1812.

[24] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21, 1953.

[25] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice-Hall, Upper Saddle River, NJ, 2004.

[26] R. E. Neapolitan and X. Jiang. *Probabilistic Methods for Financial and Marketing Informatics*. Morgan Kaufmann, San Francisco, CA, 2007.

[27] J. Hernández Orallo and C. Ferri Ramírez y J. Ramírez Quintana. *Introducción a la minería de datos*. Pearson-Prentice Hall, Madrid, 2004.

[28] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[29] J. Pearl. *Causality. Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK, 2000.

[30] M. Ramoni and P. Sebastiani. Learning Bayesian networks from incomplete databases. Technical report KMI-TR-43, Knowledge Media Institute, The Open University, 1996.

[31] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 17(2):461–464, 1978.

[32] A. Shalloway and J. R. Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley, Boston, MA, 2004.

[33] P. Spirtes, C. Glymour, and R. Scheines. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9:62–72, 1991.

[34] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. The MIT Press, Cambridge, Massachusetts, second edition, 2000.

[35] P. Spirtes and C. Meek. Learning Bayesian networks with discrete variables from data. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 294–300, Menlo Park, CA, 1995. AAAI Press.

[36] I. Stewart. *De aquí al infinito: las matemáticas de hoy*. Editorial Crítica (Grupo Planeta), Barcelona, 1998.

[37] O. Wilde. *The importance of being Earnest*. Courier Dover Publications, 1990.