Facultad de **Ciencias**

Cálculo eficiente de SHAP para modelos de redes bayesianas gaussianas

Efficient computation of SHAP values for gaussian bayesian network models

Trabajo de Fin de Máster

para acceder al

Máster en Matemáticas y Computación

Autor: Jose Ramón Buergo Lagunas

Tutor: Camilo Palazuelos Calderón

Septiembre- 2025

Resumen

Este Trabajo de Fin de Máster estudia la relación entre la teoría de juegos y la inteligencia artificial explicable (XAI), con un enfoque específico en el cálculo de los valores de Shapley en redes bayesianas gaussianas (RBGs). Aunque estos valores ofrecen un marco sólido para atribuir la contribución de cada variable, su cálculo suele ser inviable debido al gran número de coaliciones necesarias. El objetivo principal es aprovechar las propiedades estructurales y estadísticas de las RGBs para proponer un marco teórico y computacional que permita un cálculo exacto y eficiente de los valores de SHAP. A través del uso de entornos de Markov y de la estructura local de las distribuciones condicionales, se logra reducir de manera significativa la complejidad del problema, haciéndolo escalable a modelos de mayor tamaño.

El documento se organiza en cuatro capítulos. El Capítulo 1 presenta y situa el problema dentro del ámbito del inteligencia artificial explicable. El Capítulo 2 desarrolla los fundamentos teóricos sobre las redes bayesianas gaussianas, describiendo sus bases probabilísticas, propiedades clave y principios de inferencia, con especial atención a las independencias condicionales. El Capítulo 3 constituye la aportación principal, al apoyarse en la representación canónica de las RBGs y en sus CPDs lineales gaussianas junto con los entornos de Markov para establecer una solución cerrada de los valores de SHAP. Además, se muestra que las variables irrelevantes reciben contribuciones nulas y se propone un algoritmo eficiente basado en la forma canónica y la estructura local de las CPDs. Este procedimiento evita tanto la evaluación sobre múltiples coaliciones como la inversión de grandes matrices de covarianzas, ofreciendo una solución computacionalmente viable. El Capítulo 4 ilustra la implementación práctica del marco propuesto mediante una serie de experimentos con datos reales y sintéticos. Los conjuntos de datos de Wine Quality (tinto y blanco) se utilizan para mostrar la interpretabilidad en escenarios reales, mientras que las redes sintéticas sirven para validar empíricamente las cotas de complejidad teórica.

En conclusión, este trabajo demuestra que SHAP en redes bayesianas gaussianas pueden obtenerse de un modo eficiente gracias a la explotación de sus propiedades estructurales. El método propuesto no solo confirma los resultados teóricos, sino que también ofrece una estrategia computacionalmente viable para aplicar explicaciones basadas en Shapley en la práctica, incluso en modelos de mayor escala.

Summary

This Master's Project explores the connection between game theory and explainable artificial intelligence (XAI), focusing on the computation of Shapley values in Gaussian Bayesian Networks (GBNs). Shapley values provide a principled framework for attributing the contribution of each variable, but their computation is often prohibitive due to the large number of coalitions required. The main goal of this work is to exploit the structural and statistical properties of GBNs to establish a theoretical and computational framework that enables the exact and efficient calculation of Shapley values. By leveraging the concept of Markov blankets and the local structure of conditional distributions, this approach significantly reduces the complexity of the problem, making the method scalable to realistic scenarios.

The document is organized into four chapters. Chapter 1 introduces the motivation of the study, situating the problem within explainable machine learning and highlighting the limitations of current approaches to Shapley value computation. It also outlines the objectives and the methodological perspective adopted. Chapter 2 provides the theoretical background on Gaussian Bayesian networks, describing their probabilistic foundations, key properties, and inference principles, with emphasis on conditional independencies and the advantages of the multivariate normal assumption. Chapter 3 constitutes the central contribution, building on the canonical representation of GBNs and exploiting their linear Gaussian CPDs alongside the properties of Markov blankets to establish a closed-form solution for Shapley values. In this framework, SHAP is introduced for Gaussian Bayesian Networks, showing that irrelevant variables receive null contributions and deriving an efficient algorithm based on a key identity of the canonical form and the local CPD structure. This procedure avoids costly evaluation over multiple coalitions and the inversion of the full covariance matrix, thereby providing a tractable solution. Chapter 4 illustrates the practical implementation of the framework through a series of experiments on both real and synthetic data. The Wine Quality datasets (red and white) are used to showcase interpretability in real-world scenarios, while synthetic networks serve to empirically validate the theoretical complexity bounds.

In conclusion, this work shows that Shapley values in Gaussian Bayesian Networks can be obtained exactly and efficiently by exploiting their structural properties. The proposed method not only confirms the theoretical results but also offers a computationally viable strategy for applying Shapley-based explanations in practice, even for larger-scale models.

Índice general

Capítulo 1

Introducción de conceptos

1.1. Introducción a la teoría de juegos

La palabra *juego* esta asociada a una actividad lúdica en la que participan un grupo de personas que deben cumplir con ciertas normas. Con el objetivo de ganar, estas tratarán de desarrollar una estrategia que favorezca sus opciones.

Para formalizar este enfoque y ofrecer herramientas matemáticas para analizar y prever los comportamientos estratégicos según el contexto, surge la rama de las matemáticas conocida como la teoría de juegos. El objetivo es establecer una base en la que sustentar las estrategias en situaciones donde el resultado de un participante no depende únicamente de sus propias acciones, sino también de las decisiones de otros.

Su inicio se podría considerar en 1944 con la publicación de *Game Theory and Economic Behavior*, de Von Neumann y Morgenstern, donde se introdujeron los juegos en forma estratégica y se estableció un marco matemático para analizar el comportamiento racional en situaciones competitivas. Este desarrollo se apoyó en trabajos previos de Zermelo (1913) y Borel (1921). Posteriormente, en 1950, Nash amplió la teoría introduciendo el concepto de equilibrio de Nash, una solución general para juegos no cooperativos en la que ningún jugador tiene motivos para cambiar su estrategia de manera unilateral.

La teoría de juegos abarca una amplia variedad de temas fundamentales en el estudio de la toma de decisiones estratégicas, basándose en las reglas dadas, la información que poseen los jugadores y su racionalidad. Existen tres formas principales:

- **Forma extensiva.** Vemos el juego como un árbol de decisiones; resulta útil cuando los jugadores van tomando decisiones en distintos momentos.
- **Forma estratégica.** Usaremos una matriz de pagos, donde se detallan las diferentes estrategias y recompensas de cada jugador cuando sus decisiones son simultáneas.
- **Forma coalicional.** Nos centramos en juegos donde se abre la posibilidad de cooperar, permitiendo analizar cómo los jugadores pueden agruparse y distribuir los beneficios obtenidos.

Dentro de estas encontramos herramientas para comprender distintos tipos de interacciones estratégicas, abordando conceptos como la competencia y la cooperación, el equilibrio de Nash y la influencia de la información y la incertidumbre en la toma de decisiones. En libros como [6] y [11], podemos profundizar en estos y otros fundamentos de la teoría de juegos.

Gracias a su capacidad para modelar situaciones de interacción estratégica, la teoría de juegos tiene aplicaciones en diversos campos. En economía, sirve para modelar la competencia entre empresas o la toma de decisiones en mercados complejos; en biología, para el estudio de la evolución de especies y el comportamiento de un ecosistema; y en política, para analizar las estrategias de negociación y el voto. Además, también encuentra aplicaciones en la toma de decisiones en informática y en otros muchos campos en los cuales las interacciones estratégicas estén presentes.

1.1.1. Definiciones y soluciones preliminares en juegos de N jugadores

A continuacion, vamos desarrollar concentos clave que nos ayudarán en el desarrollo y comprensión de las cuestiones que trataremos.

Definición 1. Un juego de N **jugadores** consta de una serie de eventos que puede tener un número de resultados finito. Los acontecimientos del juego dependen de la libre decisión del conjunto de jugadores $J = \{1, 2, \ldots, n\}$. En algunos casos el resultado puede depender del azar. Para cada evento, se sabe que el jugador i determina su resultado conforme a sus estrategias disponibles y cuál es su estado de información respecto al resto de jugadores. Finalmente, tras el resultado obtenido a partir de los eventos se asigna un pago a cada uno de los jugadores.

Definición 2. Un juego en forma coalicional [11] o en forma de función característica con utilidades transferibles es un modelo matemático en el que los jugadores pueden cooperar entre sí para maximizar sus posibles ganancias individuales. Este consiste en:

- Un conjunto finito de jugadores $J = \{1, 2, \ldots, n\}$.
- Una función característica, que asocia a cada subconjunto $S \in \mathscr{P}(J)$ (o coalición) un número real $v(S)$ (valor de coalición), siendo $v(\emptyset) = 0$.

Por tanto, $G = (J, v)$ es un juego en forma coalicional o en forma de función característica con utilidades transferibles si J y v están especificados.

Definición 3. Si G es un juego coalicional para los jugadores $1, 2, \ldots, n$, entonces un vector de pagos $(x_1, \ldots, x_n)$ con x_i correspondiente al jugador i es una **imputación** [11] si cumple las siguientes propiedades:

1. Racionalidad individual: $x_i \geq v(\{i\})$ para todo $1 \leq i \leq n$.
2. Racionalidad grupal: $x_1 + x_2 + \ldots + x_n = v(J)$

Denominaremos $I(J, v)$ al conjunto de imputaciones.

El vector de la pagos expresa la recompensa que recibe cada jugador, es decir, la componente x_i representa el pago que recibe el jugador i.

1.1.2. El valor de Shapley

En esta sección, nos centraremos en otra manera que tiene la teoría de juegos de afrontar la búsqueda de soluciones para los problemas coalicionales: el valor de Shapley. Este fue propuesto por Lloyd Shapley en 1953 [12] y su objetivo es proporcionar un método que represente justamente la participación de cada jugador en las posibles coaliciones para llegar a una única asignación de pagos. El valor de Shapley se preocupa por la equidad y justicia en la distribución, asegurándose de que cada jugador reciba una recompensa proporcional a la contribución marginal que aporta en todas las posibles coaliciones.

El valor de Shapley se caracteriza por ser el único valor con una serie de propiedades deseables, como la eficiencia, la simetría, la nulidad y la aditividad, que lo convierten en una solución estable y ampliamente utilizada en diversos campos, desde la economía hasta la inteligencia artificial.

La función de asignación de pagos $\phi(v)$ debe cumplir los siguientes axiomas de Shapley, como se describen en [11]:

1. **Eficiencia.** La función de asignación $\phi(v)$ debe distribuir el pago total del juego. Es decir, debe ser:
$$\sum_{i=1}^{n} \phi_i(v) = v(J)$$

2. **Simetría.** Para cualquier par de jugadores que realicen aportaciones equivalentes para cada coalición, es decir, tales que cumplan que
$$v(S \cup \{i\}) = v(S \cup \{j\}), \quad \forall S \in \mathscr{P}(J), \text{ con } i, j \notin S$$
debe ser $\phi_i(v) = \phi_j(v)$.

3. **Tratamiento del jugador pasivo.** Si un jugador no aporta ningún beneficio adicional al resto de jugadores no debe recibir ningún pago adicional. Es decir, para cada jugador $i \in J$, para el cual se verifica que
$$v(S) = v(S - \{i\}) + v(\{i\}), \text{ para toda coalición } S \text{ con } i \in S$$
debe ser $\phi_i(v) = v(\{i\})$.

4. **Aditividad.** La función de asignación ϕ debe ser invariante a cualquier descomposición arbitraria del juego. Formalmente, dados dos juegos cualesquiera (J, v_1) y (J, v_2) debe ser:
$$\phi(v_1 + v_2) = \phi(v_1) + \phi(v_2)$$

La única imputación que verifica que se cumplen estos cuatro axiomas se conoce como el valor de Shapley.

Definición 4. Valor de Shapley ($\phi(v)$). [11] Dado un juego coalicional de n jugadores, daremos una regla para determinar la imputación $\phi(v)$=$(\phi_1(v),\phi_2(v),\ldots,\phi_n(v))$ que verifica los axiomas de Shapley.

$$\phi_i(v) = \sum_{S\in\mathscr{P}(J),\ S\neq\emptyset} q(s)[v(S)-v(S-\{i\})]$$

en donde $q(s)=\frac{(s-1)!(n-s)!}{n!}$, siendo $s=|S|$, el número de jugadores que hay en la coalición S.

El valor de Shapley considera todas las maneras de ordenar los n jugadores y, para cada una de estas, examina el valor que aporta añadir el jugador i en las diferentes posiciones. Si S es el conjunto de jugadores antes de añadir el jugador i, su valor de adhesión sería $v(S\cup\{i\})-v(S)$. El pago $\phi_i(v)$ que se calcula con la fórmula anterior representa este valor medio de adhesión a todas las posibles coaliciones.

Proposición 1.1. El valor de Shapley definido anteriormente pertenece al conjunto de imputaciones.

Demostración: Empecemos verificando que se cumple la racionalidad individual ($x_i \geq v(\{i\})$, $1\leq i\leq n$). Sea i un jugador cualquiera y $S\in\mathscr{P}(J)$ una coalición que no contiene a nuestro jugador i. Si tratamos de calcular cuál es el valor que añade este jugador a la coalición obtenemos $v(S\cup\{i\})-v(S)\geq v(\{i\})$. Como x_i es la media de estos valores para este tipo de coaliciones, se obtiene el resultado que buscábamos.

Continuemos ahora con la racionalidad grupal ($x_1+\ldots+x_n=v(\{j\})$).

Si empezamos desde una coalición vacía y vamos añadiendo jugadores de uno en uno, calculando el valor que aporta cada uno hasta llegar al número total de jugadores obtenemos lo siguiente:

$$v(\{1\})+[v(\{1,2\})-v(\{1\})]+[v(\{1,2,3\})-v(\{1,2\})]+\ldots+[v(\{1,\ldots,n\})-v(\{1,\ldots,n-1\})]=$$
$$[v(\{1\})+v(\{1,2\})+\ldots+v(\{1,\ldots,n\})]-[v(\{1\})+v(\{1,2\})+\ldots+v(\{1,\ldots,n-1\})]=v(\{1,\ldots,n\})$$

Indepedientemente del orden en el que añadamos los jugadores, llegamos a que el valor total es $v(J)$.Por lo tanto, $x_1+\ldots+x_n$, que es la suma de los valores promedio agregados por los jugadores, será igual a $v(J)$, como se deseaba. □

Ejemplo. Durante unas elecciones a las cuales se presentaban cuatro candidatos $\{1,2,3,4\}$, se han obtenido los siguientes porcentajes de votos $\{35,35,20,10\}$ correspondientes a cada uno, respectivamente. A continuación, calcularemos los valores de Shapley para cuantificar la importancia que refleja Shapley respecto a los resultados que ha obtenido cada candidato. Para ello, definiremos la siguiente función de pago:

$$f(S)=\begin{cases}1, & \text{si } \sum_{i\in S} p_i > 50\\ 0, & \text{si } \sum_{i\in S} p_i \leq 50\end{cases}$$

Esta función asigna 1 a las coaliciones con mayoría absoluta y 0 a las que no.

$$\phi_1(f) = \frac{1}{4}f(\{1\}) + \frac{1}{12}[f(\{1,2\}) - f(\{2\})] + \frac{1}{12}[f(\{1,3\}) - f(\{3\})] + \frac{1}{12}[f(\{1,4\}) - f(\{4\})] + \frac{1}{12}[f(\{1,2,3\}) - f(\{2,3\})] + \frac{1}{12}[f(\{1,2,4\}) - f(\{2,4\})] + \frac{1}{12}[f(\{1,3,4\}) - f(\{3,4\})] + \frac{1}{4}[f(\{1,2,3,4\}) - f(\{2,3,4\})] = 0 + \frac{1}{12} + \frac{1}{12} + 0 + 0 + \frac{1}{12} + \frac{1}{12} + 0 = 1/3$$

De manera análoga, obtenemos que $\phi(f) = (\phi_1(f), \phi_2(f)\phi_3(f), \phi_4(f)) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$.

En este caso, los resultados del valor de Shapley le asignarían la misma importancia a los tres primeros candidatos, aunque el tercero tenga un menor porcentaje que los dos primeros. A su vez, se observa que el cuarto candidato no tiene importancia, a pesar de que su porcentaje no es nulo. Por otro lado, en este ejemplo podemos observar cómo se cumple la propiedad de eficiencia $x_1 + x_2 + x_3 + x_4 = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 0 = 1 = f(1,2,3,4)$. Además, vemos que también se cumple la propiedad de simetría con los jugadores $\{1\}$ y $\{2\}$, y la propiedad de pasividad con el jugador $\{4\}$. En este caso, los resultados del valor de Shapley le asignarían la misma importancia.

1.2. Introducción a los modelos de inteligencia artificial

Durante las últimas décadas, la generación de datos ha crecido de manera exponencial. Esta gran cantidad de información se ha convertido en un recurso valioso que, si se trata de forma inteligente, puede fomentar grandes avances en múltiples campos. Con el propósito de aprovechar estas ventajas, surge la inteligencia artificial (IA) o el aprendizaje automático (en inglés, *machine learning [ML])*.

La *IA* es una rama de la informática que busca crear sistemas capaces de realizar tareas que normalmente requieren inteligencia humana, como el aprendizaje, la toma de decisiones y la resolución de problemas. El propósito principal de la IA es automatizar tareas complejas y mejorar la eficiencia en diversas aplicaciones.

Dentro de este campo, el *ML* se define como el desarrollo de métodos y algoritmos que usan los ordenadores para "*aprender"* a partir de un conjunto de datos dados, con el fin de realizar una tarea, como podría ser una predicción o clasificación de un resultado.

Una de las ramas matemáticas que más ha contribuido al crecimiento de estos métodos de *machine learning* es la teoría de la probabilidad y la estadística, de la cual toma modelos prestados como los de regresión lineal en el caso de realizar predicciones o los de regresión logística para hacer clasificaciones. A su vez, esta disciplina también proporciona herramientas para verificar la calidad de los resultados.

Dependiendo de la forma en que queramos tratar los diferentes tipos de datos y el objetivo que busquemos, se suelen distinguir tres grandes enfoques dentro del aprendizaje automático:

aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Entre estos, el aprendizaje supervisado va a ser el que más nos interese en este TFM. Dentro del aprendizaje supervisado, los conjuntos de datos deben incluir tanto los datos de estudio como información sobre los resultados. El objetivo es construir un modelo a partir de comparaciones con los datos de entrada, que sea capaz de predecir correctamente la salida.

Este grupo incluye diversos algoritmos como naive Bayes, máquinas de vectores soporte, árboles de decisión o redes neuronales artificiales. Además, dentro de este grupo podemos distinguir dos grupos: los métodos de clasificación, cuyo objetivo es determinar a qué categoría pertenece una observación en función de los datos observados, y los métodos de regresión, que buscan devolver un valor numérico dado por una función continua en base a los datos de entrada.

Para profundizar en los conceptos de inteligencia artificial mencionados, se recomienda consultar el libro [7].

Ahora bien, no basta con construir un modelo que funcione sobre los datos de entrenamiento, es fundamental asegurarnos de que el modelo va a seguir rindiendo debidamente con datos no vistos. Esta evaluación debe tener en cuenta tanto los resultados obtenidos como la capacidad de verificación e interpretación del método utilizado.

En general, cuando vamos a entrenar un modelo lo hacemos de la siguiente manera. Partimos el conjunto de datos en tres partes: entrenamiento, validación y test. El mayor volumen de datos se concentra en el grupo de entrenamiento, que es utilizado para construir el modelo. Esta construcción se evalúa con los datos de validación y, finalmente, se evalúa el modelo con los datos de prueba. Con esto conseguimos minimizar el riesgo de que nuestro modelo pueda ser demasiado sensible, es decir, que la modificación mínima de un dato de entrada provoque grandes cambios en el modelo o que pueda estar sobreentrenado, lo que significaría que le falta generalidad y le cueste trabajar con datos nuevos.

Otro aspecto significativo que se ha de tener en cuenta en nuestro modelo es la **interpretabilidad**. Este concepto se puede definir como la capacidad que tiene el ser humano de entender los resultados obtenidos mediante el machine learning, así como sus causas.

Cuanto mayor sea la interpretabilidad que posee un modelo más sencillo resultará comprender su comportamiento de manera que pueda llegar a ser explicado. Para aumentar el nivel de confianza que una persona le pueda otorgar al modelo, dependemos de este concepto para asegurar la presencia de los siguiente rasgos:

- **Equidad e imparcialidad** a la hora de hacer predicciones.
- **Fiabilidad**, para que pequeños cambios no afecten considerablemente la predicción.
- **Causalidad**, para que se pueda asociar una causa al resultado.

Algunos métodos, como los árboles de decisión, los modelos de regresión o los modelos lineales generalizados, son considerados modelos interpretables debido a su origen estadístico,

que les aporta características que permiten evaluar la precisión que tiene un modelo y sus resultados. Por el contrario, tenemos los denominados agnósticos del modelo, que presentan un mayor poder predictivo a costa de la interpretabilidad. En estos casos, es necesario que vayan acompañadas de herramientas como [10] o algoritmos relacionados con el valor de Shapley, que ayudan a poder determinar la contribución de cada variable.

La importancia de las variables dentro de un modelo de aprendizaje automático influye en la precisión y la generalización que se puede hacer sobre un modelo. Con esto nos referimos al valor que aporta cada variable para llegar al resultado y comprender mejor su funcionamiento. Considerar la importancia de las variables es crucial para interpretar y confiar en los resultados del modelo, lo que nos permite comprender por qué toma ciertas decisiones.

Con los conocimientos introducidos sobre aprendizaje automático podemos hablar de uno de los problemas que ha surgido dentro de este campo en **los modelos de caja negra**. Estos se caracterizan por tener una estructura interna compleja y difícil de interpretar. Es cierto que este tipo de modelos puede lograr conseguir unos resultados muy favorables, pero, por otro lado, su falta de interpretabilidad puede ser una gran desventaja.

En relación con el tema anterior, el auge del aprendizaje automático durante estos últimos años, que lo ha hecho estar presente en múltiples campos, ha evidenciado la necesidad de regular su aplicación. Aunque en algunos ámbitos, como el sistema de recomendación de películas de una plataforma, no sea relevante una explicación, en otros casos, especialmente en el terreno económico o de salud, esto si atesora gran importancia. Por esta razón, esta falta de transparencia puede desvirtuar la confianza en un modelo y plantear preocupaciones éticas y legales sobre su uso. Actualmente, podemos ver que en Europa se esta trabajando en su regulación [5].

1.3. Inteligencia artificial explicable

Debido a lo introducido anteriormente, se comienza el desarrollo de la siguiente rama, la **inteligencia artificial explicable (XAI)** [2] y [4]. Esta se centra en la producción de detalles y razones que nos ayuden a que nuestro modelo sea más sencillo de entender. Esto puede implicar que para lograr una mayor interpretabilidad del modelo en busca de una explicación coherente sea necesario sacrificar algo de rendimiento. Con este factor explicatorio, podemos lograr ventajas tanto a nivel de resultados como a nivel de modelos, identificando las debilidades que muestran y debemos mejorar.

Dos conceptos que habitualmente se confunden en este ámbito, aunque recogen dos conceptos diferentes, son:

- **Interpretabilidad**, que hace referencia a la capacidad de un modelo para ser comprendido por humanos y permitir que estos entiendan directamente cómo llega a una decisión.

- **Explicabilidad**, que se refiere a la capacidad del sistema para proporcionar descripciones o razonamientos sobre sus decisiones, incluso cuando el modelo con el que se trabaja no disponga de herramientas propias para su interpretabilidad. En estos casos, la explicación no proviene del modelo en sí, sino de un proceso complementario.

Antes de profundizar, vamos a dar una definición más formal sobre la XAI.

> *"Dado un público, una Inteligencia Artificial Explicable es aquella que produce detalles o razones para hacer su funcionamiento claro o fácil de entender."*

Esta definición nos subraya dos aspectos clave, que la explicación puede variar según a quién este dirigida y que el objetivo principal es facilitar la comprensión del funcionamiento de los modelos.

Desde esta perspectiva, podemos responder a las tres grandes preguntas que articulan los objetivos de la XAI:

- **¿Por qué?** Los sistemas de IA están cada vez más presentes en procesos críticos, para superar las barreras entre investigación y sectores empresariales, y porque enfocarse únicamente en el rendimiento puede limitar su utilidad.
- **¿Para qué?** Para mejorar la transparencia, fortalecer la confianza de los usuarios y contribuir al desarrollo de sistemas más seguros, justos y robustos.
- **¿Cómo?** A través de técnicas que permitan descomponer, simular o aproximar el comportamiento del modelo e incorporando explicaciones textuales, visuales o locales.

A partir de estos objetivos generales, se distinguen cuatro niveles de actuación, cada uno con técnicas y metas particulares.

- **Explicabilidad de los datos**: Esto engloba las técnicas aplicadas sobre nuestro conjunto de datos inicial para conseguir una descripción detallada de los mismos.
- **Explicabilidad del modelo**: Este término se refiere a la capacidad de los modelos para ofrecer una comprensión propia o la utilización de herramientas que faciliten esta comprensión.
- **Explicabilidad de los resultados**, podría considerarse uno de los aspectos más importantes dentro de este campo y se encarga del tratamiento de métodos y algoritmos que buscan explicar el modelo una vez obtenido los resultados.
- **Evaluación de las explicaciones**, se enfoca en medir cúan adecuada es la explicación en relación con su objetivo y a quien se dirige.

En el marco de la inteligencia artificial explicable (XAI), uno de los aspectos fundamentales para llegar a obtener interpretaciones fiables y útiles de los modelos de aprendizaje automático

reside en la estrategia adoptada. Habitualmente, pueden distinguirse dos metodologías: los modelos intrínsecamente explicables, también denominados transparentes, y las técnicas post-hoc, aplicadas a modelos opacos tras su entrenamiento.

Modelos transparentes

Su diseño ya conlleva cierto grado de explicabilidad, y su estructura interna y los mecanismos mediante los cuales se toman las decisiones permiten un análisis directo sin recurrir a herramientas externas. En esta categoría se incluyen modelos clásicos y bien establecidos que permiten un análisis interpretativo natural:

- **Regresión lineal**: permite interpretar de forma directa el impacto de cada variable de entrada a través de los coeficientes del modelo.
- **Árboles de decisión**: Presenta una forma ramificada basándose en características de los datos. Su alta interpretabilidad radica en su estructura jerárquica y en sus reglas de decisión claras, que facilitan seguir cada paso lógico hasta el resultado final.
- **K-Nearest Neighbors (K-NN)**: realiza las predicciones basándose en la similitud con los "k" ejemplos más cercanos en el espacio de datos. Se caracteriza por intentar identificar patrones locales, aunque depende críticamente de la elección de la "k".
- **Aprendizaje basado en reglas**: generan conjuntos de reglas explícitas que describen con precisión el razonamiento detrás de las decisiones del modelo.
- **Modelos aditivos generalizados (GAMs)**: amplían la regresión lineal permitiendo la incorporación de funciones no lineales sobre cada variable.
- **Modelos bayesianos**: destacan por su capacidad para modelar incertidumbre y relaciones condicionales entre variables.

El uso de este tipo de modelos probabilísticos resulta especialmente interesante cuando se requieren explicaciones individualizadas. Además, en algunos casos podemos aprovechar sus propiedades estructurales para desarrollar enfoques analíticos eficientes.

Técnicas post-hoc

Cuando se trabaja con modelos de alto rendimiento pero escasa interpretabilidad, como las redes neuronales profundas, resulta habitual recurrir a técnicas que permitan explicar sus predicciones sin modificar su estructura. El objetivo es incorporar descripciones comprensibles del comportamiento del modelo, ya sea a nivel global o para predicciones individuales. Entre las herramientas más utilizadas se encuentran:

- **LIME (Local Interpretable Model-Agnostic Explanations)**: genera modelos interpretables alrededor de una instancia concreta.
- **SHAP (SHapley Additive exPlanations)**: se basa en la teoría de juegos cooperativos, otorga a cada característica un valor estimado que representa su impacto marginal esperado sobre la predicción.

- **PDPs (Partial Dependence Plots)**: permiten visualizar el efecto marginal de una o varias variables sobre la predicción, manteniendo las demás constantes.
- **Contrafactuales**: buscan dar ejemplos mínimos que modificarían la salida del modelo.
- **Técnicas visuales**: destacan regiones de la entrada que han influido significativamente en la activación del modelo.

Entre las técnicas post-hoc, SHAP destaca por su solidez teórica al trasladar conceptos de la teoría de juegos y por poder ofrecer explicaciones locales y globales. Aunque su aplicación práctica plantea ciertos desafíos computacionales.

1.4. Objetivos y estructura del trabajo

1.4.1. Objetivos del trabajo

Una vez sentadas las bases conceptuales, es necesario precisar la finalidad de este Trabajo de Fin de Máster. El objetivo principal es aprovechar las propiedades estructurales y estadísticas de las redes bayesianas gaussianas para establecer un marco teórico y práctico que permita el cálculo exacto y eficiente de los valores SHAP. A diferencia de los enfoques generales, que requieren evaluar el modelo en un gran número de coaliciones y resultan computacionalmente costosos, se muestra que, bajo ciertos supuestos probabilísticos y gracias al concepto de entorno de Markov, es posible obtener un método que reduce significativamente la complejidad del problema. Sobre esta base, se desarrolla e implementa un procedimiento computacional con el fin de mostrar su viabilidad.

1.4.2. Estructura del documento

El resto de este documento se organiza en los siguientes capítulos:

- **Capítulo 2. Redes bayesianas gaussianas.** Presenta el marco teórico necesario sobre redes bayesianas gaussianas, sus propiedades fundamentales y los principios de inferencia probabilística.
- **Capítulo 3. SHAP en redes bayesianas gaussianas.** Expone los resultados teóricos principales del trabajo, incluyendo los teoremas sobre los valores SHAP, el papel del entorno de Markov y la derivación de un procedimiento computacionalmente eficiente para su cálculo.
- **Capítulo 4. Ejemplo práctico.** Apoya los resultados obtenidos en el enfoque desarrollado en el capítulo anterior. Implementando su aplicación sobre un conjunto de datos reales y realizando una experimentación con datos sintéticos, lo que permite analizar tanto la interpretabilidad como la complejidad computacional.

Capítulo 2

Redes bayesianas gaussianas

2.1. Fundamentos estadísticos básicos

Antes de entrar en el análisis de las redes bayesianas gaussianas [8]||[9], es útil detenernos brevemente en ciertos conceptos estadísticos que nos permitirán entender mejor su construcción y comportamiento.

Definición 5. Un espacio de probabilidad es una terna $(\Omega, \mathscr{F}, \mathbb{P})$ que formaliza un experimento aleatorio:

- Espacio muestral (Ω) : Conjunto de todos los resultados posibles.
- σ-álgebra ($\mathscr{F}$) : Familia de subconjuntos de Ω llamados eventos, incluyendo Ω.
- Medida de probabilidad ($\mathbb{P}$): Función $\mathbb{P} : \mathscr{F} \to [0,1]$ que cumple:
  - A cada evento A le corresponde un número real tal que $0 \leq \mathbb{P}(A) \leq 1$.
  - Necesariamente, $\mathbb{P}(\emptyset) = 0$ y $\mathbb{P}(\Omega) = 1$
  - Para eventos disjuntos $\{A_n\}_{n=1}^{\infty} \subset \mathscr{F}$, se cumple que $\mathbb{P}(A_1 \cup A_2 \cup \ldots) = \sum_{n=1}^{\infty} \mathbb{P}(A_n)$.

Definición 6. Una variable aleatoria es una función medible $X : \Omega \to \mathbb{R}$ que asigna valores numéricos a los resultados de Ω.

Una vez definidos los conceptos de espacio de probabilidad y variable aleatoria, es fundamental describir los tipos de distribuciones de probabilidad. A grandes rasgos, distinguimos entre distribuciones discretas y continuas, dependiendo del conjunto de valores que puede tomar la variable aleatoria.

Definición 7. Una variable aleatoria continua toma valores en un subconjunto no numerable del espacio real. En este caso, la probabilidad de que tome un valor exacto es nula, es decir, $\mathbb{P}(X = x) = 0$ para todo x. La distribución se describe mediante una función de densidad de probabilidad $f_X(x)$, tal que:

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f_X(x)\, dx$$

Los ejemplos clásicos son la uniforme, la normal (gaussiana), la exponencial y la gamma.

Dentro de las distribuciones continuas, la distribución normal sobresale por su versatilidad y propiedades matemáticas. Su relevancia teórica y práctica extendida al caso multivariante la convierte en la base natural para modelar sistemas complejos.

Definición 8. Una variable aleatoria continua sigue una distribución normal univariada si su función de densidad de probabilidad está parametrizada por una media $\mu \in \mathbb{R}$ y una varianza $\sigma^2 > 0$, y está dada por:

$$X \sim \mathcal{N}(\mu, \sigma^2), \quad \text{o bien} \quad p(x) = \mathcal{N}(x \mid \mu, \sigma^2)$$

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \quad \Rightarrow \quad \mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sigma}\,\varphi\left(\frac{x-\mu}{\sigma}\right)$$

donde $\varphi(x)$ es la densidad de la normal estándar.

Hasta ahora, nos hemos centrado en el caso univariante. Sin embargo, muchos fenómenos requieren analizar la relación entre varias variables aleatorias al mismo tiempo. Para poder abordar este tipo de problemas, recurriremos a las distribuciones conjuntas, marginales y condicionales, que nos permiten descomponer y entender las dependencias entre variables.

Definición 9. Sea $\mathbf{X} = (X_1, \ldots, X_n)$ un vector de variables aleatorias continuas. Su distribución conjunta está definida mediante una función de densidad $f(\mathbf{x})$ que cumple:

$$f(\mathbf{x}) \geq 0 \;\; \forall x \in \mathbb{R}^n, \quad \int_{\mathbb{R}^n} f(\mathbf{x})\, d\mathbf{x} = 1.$$

Definición 10. La distribución marginal de una variable continua X_i se obtiene integrando la densidad conjunta respecto a las demás variables. Esta operación permite centrarnos en el comportamiento individual de X_i, ignorando la presencia de otras variables en el sistema.

$$f(x_i) = \int f(x_i, \mathbf{x}_{-i})\, d\mathbf{x}_{-i},$$

donde $\mathbf{x}_{-i}$ denota el conjunto de todas las variables excepto X_i.

Definición 11. La distribución condicional describe el comportamiento de una variable continua partiendo de que conocemos el valor de otra. Para dos variables aleatorias X e Y, se define como:

$$f(y \mid x) = \frac{f(x,y)}{f(x)} \quad \text{si } f(x) > 0.$$

A partir del concepto de densidad condicional, se pueden derivar resultados fundamentales como:

- **La ley de probabilidad total**, que permite obtener la densidad marginal de una variable integrando sobre otra:

$$f(y) = \int f(y \mid x)\, f(x)\, dx.$$

- **La regla de Bayes** que relaciona la probabilidad condicional inversa, permitiendo poder actualizar nuestro conocimiento sobre X cuando observamos Y.

$$f(x \mid y) = \frac{f(y \mid x)\, f(x)}{f(y)}.$$

En los modelos de probabilidad, la noción de independencia entre variables permite simplificar estructuras complejas. Su generalización, la independencia condicional, no solo conserva esta utilidad, sino que permite construir representaciones eficientes del conocimiento que pueden plasmarse en forma de grafos, como ocurre en las redes bayesianas gaussianas.

Definición 12. Dos variables aleatorias continuas X e Y se dicen independientes si su densidad conjunta se factoriza como:

$$f(x,y) = f(x) \cdot f(y), \quad \forall x,y.$$

Esta condición nos dice que conocer el valor de una variable no nos aportará información adicional sobre la otra. Esto equivale a que $f(y \mid x) = f(y)$.

Definición 13. Dados tres conjuntos de variables aleatorias continuas X, Y y Z, decimos que X es condicionalmente independiente de Y dado Z si:

$$f(x,y \mid z) = f(x \mid z) \cdot f(y \mid z), \quad \text{cuando } f(z) > 0.$$

Intuitivamente, esto significa que, una vez ya sabemos el valor que toma Z, las variables X e Y no contienen información adicional entre sí, es decir que:

$$f(x \mid y,z) = f(x \mid z)$$

Por tanto, vemos que condicionar en Y no cambia la distribución de X si ya conocemos Z.

Volviendo al ejemplo clásico de distribución conjunta continua, tenemos la normal multivariante. Sean $\mathbf{X} = (X_1, \ldots, X_n)$ variables aleatorias con distribución:

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$$

Por lo que su función de densidad es:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

donde $\mu \in \mathbb{R}^n$ es el vector de medias, y $\Sigma \in \mathbb{R}^{n\times n}$ es la matriz de covarianzas, simétrica y definida positiva. Para una normal multivariante estándar tendríamos que su vector de medias μ es nulo y su matriz de covarianzas Σ es la identidad $\mathbf{I}$.

Respecto a los momentos de la distribución, el vector de medias se define como $\mu = \mathbb{E}[\mathbf{X}]$ y la matriz de covarianzas como $\Sigma = \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top$. Para los términos de sus componentes, denotaremos como X_i la i-ésima variable aleatoria, teniendo que:

- $\mu_i = \mathbb{E}[X_i]$ es la media de X_i,

- $\Sigma_{ii} = \text{Var}[X_i]$ es la varianza de X_i,
- $\Sigma_{ij} = \text{Cov}[X_i, X_j] = \mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]$ es la covarianza entre X_i y X_j.

Dado que toda matriz definida positiva es invertible, es posible emplear $J = \Sigma^{-1}$ conocida como matriz de precisión o matriz de información, para llegar a una representación alternativa de la normal multivariante. De esta forma se consigue reescribir la función de densidad con una expresión alternativa:

$$-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu) = -\frac{1}{2}(\mathbf{x}-\mu)^T J(\mathbf{x}-\mu) = -\frac{1}{2}\left[\mathbf{x}^T J\mathbf{x} - 2\mathbf{x}^T J\mu + \mu^T J\mu\right]$$

Como el último término es una constante, llegamos a:

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^T J\mathbf{x} + (J\mu)^T \mathbf{x}\right)$$

Esta formulación de la densidad gaussiana se llama forma de información, y al vector obtenido como $\mathbf{h} = J\mu$ se le denomina vector de potencial.

Esta distribución es especialmente interesante porque:

- Las marginales de una normal multivariante también son normales.
- Las condicionales también siguen distribuciones normales.
- Su estructura de covarianzas y correlaciones puede visualizarse mediante grafos, lo que da paso a las redes bayesianas gaussianas.

Por ejemplo, para la marginalización basta con seleccionar los componentes relevantes del vector de medias y la submatriz correspondiente de la matriz de covarianzas. Y si queremos condicionar, dividimos $\mathbf{X} = (\mathbf{X}_A, \mathbf{X}_B)$, teniendo que:

$$(\mathbf{X}_A \mid \mathbf{X}_B = \mathbf{x}_B) \sim \mathcal{N}(\mu_{A|B}, \Sigma_{A|B})$$

Es importante destacar que **marginalizar** es inmediato desde la forma con covarianzas, mientras que **condicionar** es directo desde la forma de información. Esta última implica la inversión de matrices, lo que puede no llegar a ser un problema para dimensiones reducidas, pero puede volverse computacionalmente costoso a medida que la dimensionalidad del sistema crece.

Otro aspecto destacable de la distribución normal multivariante es que la independencia entre componentes puede caracterizarse directamente a través de la matriz de covarianzas. Si $\Sigma_{ij} = 0$, entonces X_i y X_j son independientes. Por otro lado, las independencias condicionales no son visibles en la matriz de covarianzas, pero sí lo son en la matriz de información $J = \Sigma^{-1}$. En particular, $\mathbf{J}_{ij} = 0 \iff X_i \perp X_j \mid \mathbf{X} \setminus \{X_i, X_j\}$.

2.2. Redes bayesianas

Antes de introducir formalmente las redes bayesianas, conviene entender el problema fundamental que tratan de resolver. La representación explícita de una distribución conjunta sobre muchas variables aleatorias es, en general, costosa en términos computacionales y difícil de manejar estadísticamente. Las redes bayesianas nos van a permitir descomponer una distribución conjunta compleja en una colección de distribuciones condicionales más manejables, gracias a las relaciones de independencia e independencia condicional.

Definición 14. Una red bayesiana sobre un conjunto finito de variables aleatorias $\mathbf{X} = (X_1, \ldots, X_n)$ está definida por el par $(\mathscr{G}, \theta)$, donde:

- $\mathscr{G} = (V, E)$ es un grafo dirigido acíclico DAG, por su sigla en inglés.
  - V es el conjunto de nodos que corresponden a las variables $\mathbf{X}$.
  - E es el conjunto de aristas dirigidas que conectan los nodos.
  - El hecho de que sea un DAG implica dos propiedades fundamentales:
    - **Dirigido**: Todas las aristas tienen dirección ($X_i \to X_j$) indica que X_i es un padre de X_j, y por tanto, refleja que X_j depende condicionalmente de X_i.
    - **Acíclico**: No existen ciclos dirigidos, lo que quiere decir que no es posible empezar en un nodo y regresar al mismo siguiendo la dirección de las flechas.
- $\theta = P(X_i \mid \mathrm{Pa}(X_i))_{i=1}^n$ es un conjunto de distribuciones de probabilidad condicionales, una por cada variable X_i, dada su familia de padres $\mathrm{Pa}(X_i)$ en $\mathscr{G}$.

La estructura de estas redes conlleva propiedades que resultan especialmente útiles a la hora de modelar distribuciones complejas. En particular, la distribución conjunta sobre $\mathbf{X}$ se puede factorizar canónicamente como:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathrm{Pa}(X_i))$$

Esta factorización permite representar de forma compacta distribuciones que de otro modo requerirían un número exponencial de parámetros. El teorema de factorización de redes bayesianas establece que esta descomposición es válida si y solo si la distribución P es compatible con la estructura de independencias condicionales inducida por el grafo $\mathscr{G}$.

Para una comprensión más intuitiva de cómo las redes bayesianas capturan las relaciones de independencia e independencia condicional, el lector puede consultar un ejemplo detallado desarrollado en el Anexo I.1, basado en variables discretas, aunque en lo que sigue nos centraremos en el caso continuo, concretamente en las redes bayesianas gaussianas. Este ejemplo resulta de gran utilidad para entender el funcionamiento y la aplicabilidad de estos modelos.

En lo que sigue, veremos cómo construir una distribución conjunta continua utilizando redes bayesianas.

Definición 15. Decimos que una variable aleatoria continua Y sigue una **distribución lineal gaussiana** condicionada a sus padres $X_1,\dots,X_k$ si su distribución condicional es normal, su media depende linealmente de los valores de sus padres y su varianza es constante.

$$p(Y \mid X_1 = x_1,\dots,X_k = x_k) = \mathcal{N}\left(\beta_0 + \sum_{j=1}^{k} \beta_j x_j, \sigma^2\right)$$

Si $\mathbf{x} = (x_1,\dots,x_k)^T$ y $\beta = (\beta_1,\dots,\beta_k)^T$, su media condicional puede escribirse en forma vectorial como:

$$\mathbb{E}[Y \mid \mathbf{x}] = \beta_0 + \beta^T \mathbf{x}$$

Esta distribución se corresponde con una normal univariante y además es análoga al modelo de regresión lineal clásica con ruido gaussiano. Los coeficientes β_j reflejan la influencia lineal de cada variable X_j sobre la variable dependiente Y, mientras que el parámetro σ^2 representa la varianza del término de error.

Definición 16. Una **red bayesiana gaussiana** es una red bayesiana donde todas las variables son continuas y todas las distribuciones condicionales son lineales gaussianas.

Un resultado clave es que estas redes constituyen una representación alternativa para la clase de distribuciones normales multivariantes. Esto significa que, si construimos una red bayesiana con distribuciones condicionales lineales gaussianas, la distribución conjunta sobre todas las variables será una normal multivariante. Y al revés, cualquier distribución normal multivariante puede representarse como una red bayesiana gaussiana.

Con esto se concluye que, dada Y, una variable con modelo lineal gaussiano respecto a sus padres $X_1,\dots,X_k$, asumiendo que $\mathbf{X} = (X_1,\dots,X_k)$ sigue una distribución conjunta $\mathcal{N}(\mu,\Sigma)$. Entonces:

- La distribución marginal de Y es también gaussiana, con:

$$\mu_Y = \beta_0 + \beta^\top \mu, \qquad \sigma_Y^2 = \sigma^2 + \beta^\top \Sigma \beta$$

- La distribución conjunta $(\mathbf{X}, Y)$ es una distribución normal multivariante, donde la covarianza entre X_i e Y viene dada por:

$$\mathrm{Cov}[X_i, Y] = \sum_{j=1}^{k} \beta_j \Sigma_{ij}$$

Con esto, se garantiza que la distribución conjunta es siempre normal multivariante.

Veamos otro resultado sobre su comportamiento, sea $(\mathbf{X}, \mathbf{Y})$ un vector aleatorio con distribución normal conjunta definido por la siguiente ecuación:

$$p(\mathbf{X}, \mathbf{Y}) = \mathcal{N}\left(\begin{pmatrix}\mu_{\mathbf{X}} \\ \mu_{\mathbf{Y}}\end{pmatrix}, \begin{pmatrix}\Sigma_{\mathbf{XX}} & \Sigma_{\mathbf{XY}} \\ \Sigma_{\mathbf{YX}} & \Sigma_{\mathbf{YY}}\end{pmatrix}\right)$$

Entonces tenemos que la densidad condicional $p(\mathbf{Y}|\mathbf{X})$ sigue una distribución $\mathcal{N}\left(\beta_0 + \beta^\top \mathbf{X}, \sigma^2\right)$ con parámetros:

$$\beta_0 = \mu_{\mathbf{Y}} - \Sigma_{\mathbf{YX}}\Sigma_{\mathbf{XX}}^{-1}\mu_{\mathbf{X}} \qquad \beta = \Sigma_{\mathbf{XX}}^{-1}\Sigma_{\mathbf{XY}} \qquad \sigma^2 = \Sigma_{\mathbf{YY}} - \Sigma_{\mathbf{YX}}\Sigma_{\mathbf{XX}}^{-1}\Sigma_{\mathbf{XY}}$$

Este resultado proporciona un mecanismo para construir redes bayesianas a partir de distribuciones gaussianas multivariantes.

De manera más general, tenemos que si suponemos que las variables $\mathbf{X} = (X_1, \ldots, X_n)$ siguen una distribución lineal gaussiana conjunta, entonces, dado un orden cualquiera de $X_1, X_2, \ldots, X_n$ sobre $\mathbf{X}$, es posible construir una red bayesiana $\mathscr{B} = (\mathscr{G}, \theta)$ tal que:

- Para cada variable X_i, sus padres en el grafo $\text{Pa}_{X_i}^{\mathscr{G}} \subseteq \{X_1, \ldots, X_{i-1}\}$.
- La distribución condicional de X_i en $\mathscr{B}$ es lineal gaussiana respecto a sus padres.

Ejemplo.

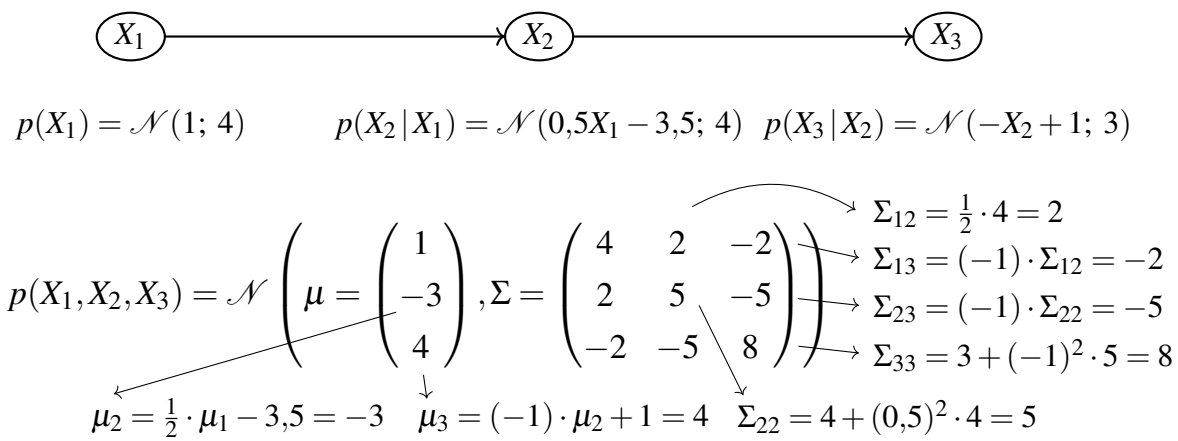


Figura 2.1: Red bayesiana con sus distribuciones condicionales, valores esperados y covarianzas.

2.3. Inferencia en redes bayesianas

Hasta ahora nos hemos visto cómo las redes bayesianas son capaces de codificar dependencias condicionales y cómo su estructura permite conseguir factorizaciones eficientes de la distribución conjunta. A continuación, abordaremos cómo aprovechar esto para realizar razonamientos probabilísticos prácticos, lo que se conoce como el problema de la inferencia.

Formalmente, dado un conjunto de variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, un subconjunto $Q \subset \mathbf{X}$ de variables objetivo *(query)* y otro subconjunto $E \subset \mathbf{X}$ de variables observadas con valores específicos $\mathbf{e}$, el problema de inferencia consiste en calcular:

$$P(Q \mid E = \mathbf{e}) = \frac{P(Q, \mathbf{e})}{P(\mathbf{e})}, \qquad \text{con } (Q \cup E) \subseteq \mathbf{X}$$

donde $P(\mathbf{e}) = \sum_Q P(Q, \mathbf{e})$ si Q es discreta, o la correspondiente integral si es continua.

A pesar de que las redes bayesianas permiten representar distribuciones conjuntas de forma compacta, esta compacidad no garantiza que el problema de inferencia sea computacionalmente sencillo. De hecho, la inferencia exacta en redes bayesianas es, en general, un problema computacionalmente intratable [13].

Como hemos venido destacando, la estructura que presentan las redes bayesianas no solo nos permite representar las distribuciones conjuntas de una manera compacta, sino que también nos habilita poder formular estrategias computacionalmente más eficientes para abordar el problema de la inferencia exacta.

Esta eficiencia proviene de la posibilidad de explotar su estructura local para poder aplicar técnicas de programación dinámica que reorganizan los cálculos, evitando así repeticiones innecesarias.

Uno de los enfoques más representativos es el algoritmo de eliminación de variables. Para poder describirlo con claridad, primero vamos a introducir algunos conceptos clave necesarios, como el concepto de factor y las operaciones básicas asociadas a su manipulación.

Un factor es una función que asigna un número real no negativo a cada combinación de valores de un conjunto de variables, y diremos que un factor tiene ámbito $X_1, \ldots, X_k$ si depende solo de esas variables.

El uso de factores en inferencia exacta requiere operaciones que permiten manipularlos, para poder combinar información local y eliminar gradualmente las variables no significativas. Estas operaciones son:

- **Producto de factores:** combina la información de distintos factores, dando lugar a uno nuevo cuyo ámbito es la unión de los anteriores.
- **Marginalización:** consiste en eliminar una variable de un factor integrando o sumando sobre todos sus valores posibles, manteniendo fijas las demás. El resultado es un nuevo factor cuyo ámbito excluye dicha variable. Es una operación conmutativa y asociativa, lo que permite flexibilidad en su aplicación.
- **Restricción:** simplifica el modelo, pues al conocer el valor que toman ciertas variables, nos permite restringirnos a estos valores concretos, eliminando las asignaciones incompatibles.

A partir de estas operaciones, podremos reorganizar la información codificada en la red para suprimir variables de forma eficiente, como veremos a continuación con el algoritmo de eliminación de variables.

En esencia, el algoritmo de eliminación de variables (VE) se basa en la idea de que podemos marginalizar variables innecesarias habiendo combinado previamente la información que las involucra. El orden en el que se realiza esta operación no afecta al resultado final, pero sí

al coste computacional, que puede crecer exponencialmente en ciertos casos, especialmente cuando se trabaja con variables discretas. Veamos un esquema general del funcionamiento de este algoritmo.

Partimos de un conjunto de factores que representa la distribución conjunta y una consulta sobre un subconjunto de variables **Y**. El algoritmo procede del siguiente modo:

1. **Identificamos las variables a eliminar.** Este será el conjunto de variables que ni pertenecen a la consulta ni están observadas $Z = X \setminus Y$.
2. **Orden de eliminación.** Se elige un orden $Z_1, \ldots, Z_k$ en el que eliminar las variables. El orden tiene un impacto decisivo sobre la eficiencia del procedimiento. Sin embargo, no hay ninguna norma establecida para su elección, habitualmente se recurre a heurísticos que guían la selección, sin que esta tenga que ser necesariamente intuitiva.
3. **Iteración por eliminación.** Para cada variable Z_i del orden elegido:
   - Detectamos de entre los factores actuales cuáles incluyen a Z_i en su ámbito.
   - Calculamos su producto, obteniendo un único factor que resume toda la información sobre Z_i.
   - Marginalizamos Z_i en este factor, añadiendo el nuevo obtenido al conjunto de factores.
   - Eliminamos del conjunto de factores los que contenían Z_i.
4. **Producto final**. Finalmente, todos los factores Z habrán sido eliminados, quedando solo los factores de nuestro interés. El producto de estos factores restantes proporciona una función sobre las variables de valor. En el caso de consultas condicionales, esta función suele requerir una renormalización.

Algoritmo 1: Eliminación de Variables (VA)

Entrada: - Conjunto de factores Φ
- Variables de consulta Y
- Orden de eliminación $Z = (Z_1, \ldots, Z_k)$

for $i = 1$ **to** k **do**
  $\Phi_{\text{rel}} \leftarrow \{\phi \in \Phi : Z_i \in \text{Scope}(\phi)\}$
  $\psi \leftarrow \prod_{\phi \in \Phi_{\text{rel}}} \phi$ ▷ Producto de factores
  $\tau \leftarrow \sum_{Z_i} \psi$ o $\tau \leftarrow \int_{-\infty}^{\infty} \psi dZ_i$ ▷ Marginalización
  $\Phi \leftarrow (\Phi \setminus \Phi_{\text{rel}}) \cup \{\tau\}$
end for
$\phi^* \leftarrow \prod_{\phi \in \Phi} \phi$
return ϕ^* ▷ Renormalizar si es necesario

Una vez expuesto el algoritmo de eliminación de variables desde un punto de vista general, el lector puede consultar el Anexo I.2, donde se presenta un ejemplo discreto que detalla paso a paso el proceso de inferencia en una red. En este se ilustra cómo se manipulan los factores y cómo el orden de eliminación afecta a la eficiencia computacional. A partir de aquí, nos centraremos en el caso continuo, explorando cómo se adapta este algoritmo al contexto de las redes bayesianas gaussianas.

Aunque en el caso continuo podría parecer, en principio, que basta con marginalizar mediante integrales y multiplicar factores como productos de funciones, lo cierto es que esto plantea desafíos importantes. En primer lugar, no existe una representación universal para los factores continuos, lo que obliga a escoger una familia paramétrica adecuada y estable frente a las operaciones de inferencia. En segundo lugar, la integración puede no ser computacionalmente viable ni admitir soluciones cerradas.

Afortunadamente, en el caso particular de las redes bayesianas gaussianas, muchas de estas dificultades desaparecen. En este caso particular, los factores pueden representarse mediante la forma canónica de la distribución normal. Esto permite adaptar las operaciones básicas de inferencia, aprovechando las propiedades del cálculo matricial para llevar a cabo los cálculos. En lo que sigue, presentaremos con detalle esta formulación y mostraremos cómo se lleva a cabo la inferencia exacta en redes gaussianas mediante una versión adaptada del algoritmo de eliminación de variables.

Definición 17. Una forma canónica sobre un conjunto de variables continuas $\mathbf{X}$ se define como una función proporcional a una exponencial cuadrática.

$$\mathscr{C}(\mathbf{X};J,h,g) = \exp\left(-\frac{1}{2}\mathbf{X}^\top J\mathbf{X} + h^\top \mathbf{X} + g\right)$$

Esta expresión se obtiene a partir de la función de densidad de la normal multivariante, donde J es la matriz de información, h es el vector de potencial y g un escalar.

$$g = -\frac{1}{2}\mu^\top J\mu - \log\left((2\pi)^{n/2}|\Sigma|^{1/2}\right)$$

La inferencia exacta en redes bayesianas gaussianas requiere tener una representación funcional que se mantenga cerrada bajo operaciones algebraicas como el producto y la marginalización. La forma canónica cumple con estos requisitos, y además permite encapsular toda la información relevante de la distribución sin necesidad de manipular directamente medias ni matrices de covarianza. Esto permite representar, de manera eficiente y compacta, las distribuciones marginales y condicionales que aparecen en estos modelos.

El producto de dos factores canónicos que comparten el mismo soporte produce un nuevo factor canónico, cuyos parámetros se obtienen simplemente sumando los respectivos términos, manteniendo asi el espacio funcional de los factores.

$$\mathscr{C}(J_1,h_1,g_1)\cdot\mathscr{C}(J_2,h_2,g_2) = \mathscr{C}(J_1+J_2,h_1+h_2,g_1+g_2)$$

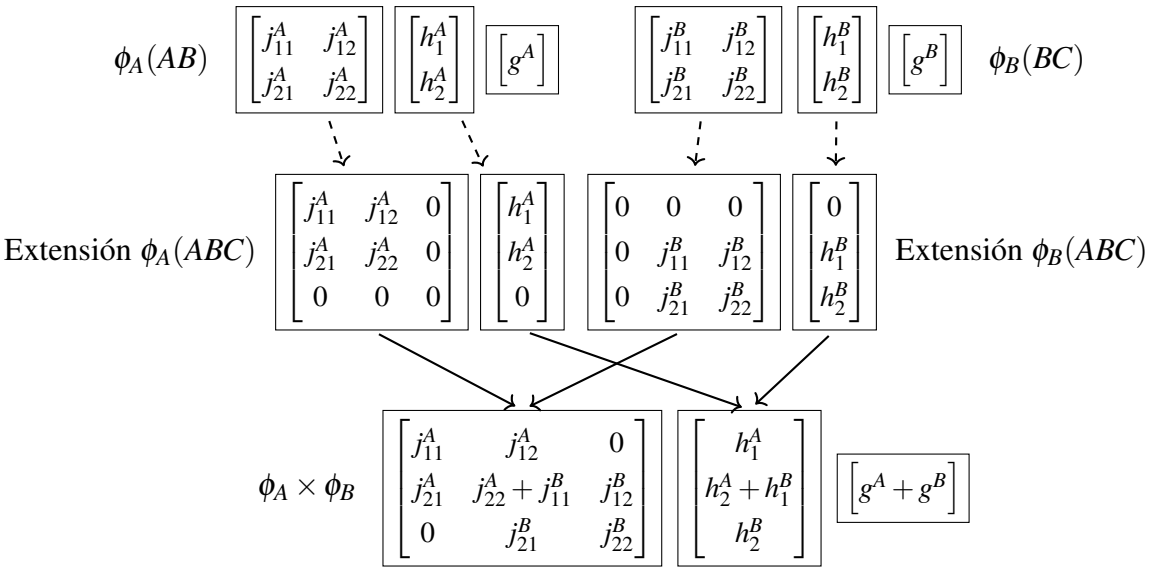


Figura 2.2: Producto de formas canónicas

De forma análoga, la división de factores se lleva a cabo restando dichos parámetros. Esta operación está definida como:

$$\frac{\mathscr{C}(J_1,h_1,g_1)}{\mathscr{C}(J_2,h_2,g_2)} = \mathscr{C}(J_1 - J_2, h_1 - h_2, g_1 - g_2)$$

Además, se puede definir un elemento neutro como $J = 0,\ h = 0$ y $g = 0$, denominado factor vacuo.

Por otro lado, tenemos la operación de marginalización, que es fundamental en el algoritmo de eliminación de variables. En la forma canónica, marginalizar un conjunto de variables $\mathbf{Y}$ en un factor definido sobre $\mathbf{X} \cup \mathbf{Y}$ equivale a integrar sobre dichas variables. Si el factor está definido como:

$$J = \begin{pmatrix} J_{\mathbf{XX}} & J_{\mathbf{XY}} \\ J_{\mathbf{YX}} & J_{\mathbf{YY}} \end{pmatrix}, \qquad h = \begin{pmatrix} h_{\mathbf{X}} \\ h_{\mathbf{Y}} \end{pmatrix}$$

entonces, si la integral existe, lo cual ocurre si $J_{\mathbf{YY}}$ es definida positiva, la marginalización de $\mathbf{Y}$ da como resultado un nuevo factor canónico sobre $\mathbf{X}$, con la siguiente forma:

$$\begin{aligned} J' &= J_{\mathbf{XX}} - J_{\mathbf{XY}} J_{\mathbf{YY}}^{-1} J_{\mathbf{YX}} \\ h' &= h_{\mathbf{X}} - J_{\mathbf{XY}} J_{\mathbf{YY}}^{-1} h_{\mathbf{Y}} \\ g' &= g + \frac{1}{2}\left(\log|2\pi J_{\mathbf{YY}}^{-1}| + h_{\mathbf{Y}}^{\top} J_{\mathbf{YY}}^{-1} h_{\mathbf{Y}}\right) \end{aligned}$$

Finalmente, la reducción de un factor por evidencia tiene una implementación eficiente en esta representación. Si fijamos un subconjunto de variables $\mathbf{Y} = \mathbf{y}$, los nuevos parámetros del factor reducido sobre $\mathbf{X}$ son:

$$J' = J_{\mathbf{XX}} \qquad h' = h_{\mathbf{X}} - J_{\mathbf{XY}}\mathbf{y} \qquad g' = g + h_{\mathbf{Y}}^{\top} - \frac{1}{2}\mathbf{y}^{\top} J_{\mathbf{YY}}\mathbf{y}$$

Todas estas operaciones que hemos ido viendo pueden realizarse en tiempo polinómico respecto al tamaño del ámbito del factor, siendo la marginalización la más costosa debido a la necesidad de invertir matrices.

Una vez vistas las operaciones básicas y cómo trabajar con la forma canónica, podemos abordar la inferencia exacta en redes bayesianas gaussianas adaptando el algoritmo de eliminación de variables. Al igual que antes, partiremos de un conjunto de factores y buscaremos calcular distribuciones marginales o condicionales sobre un subconjunto de variables.

Si queremos eliminar la variable Y, los pasos a dar son:

1. Identificar los factores que involucran a Y.
2. Multiplicar estos factores entre sí, usando debidamente $\mathscr{C}(J_1,h_1,g_1)\cdot\mathscr{C}(J_2,h_2,g_2) = \mathscr{C}(J_1+J_2,h_1+h_2,g_1+g_2)$.
3. Marginalizar la variable Y según las pautas vistas anteriormente.

Este proceso de eliminación se repite de forma iterativa, siguiendo un orden de eliminación determinado, hasta deshacerse de todas las variables no deseadas.

Cabe destacar que la condición necesaria para poder invertir la matriz $J_{\mathbf{YY}}$ queda garantizada a lo largo de todo el proceso de eliminación, siempre y cuando los factores estén bien definidos.

Cuando se dispone de evidencia, es necesario incorporar esta información reduciendo los factores que contienen variables observadas. Si una variable Z ha sido observada con valor z, y un factor está definido sobre $X \cup Z$, se debe aplicar la operación de reducción previamente descrita. Esta operación eliminará Z del dominio del factor, integrando su efecto de forma exacta y permitiendo que el resto del procedimiento de inferencia trabaje únicamente con variables no observadas.

Una ventaja clave de la representación en forma canónica es que permite realizar inferencia exacta con complejidad polinómica, ya que las operaciones elementales tienen costes cuadráticos o cúbicos respecto al número de variables implicadas. Esto contrasta favorablemente con el crecimiento exponencial que presentan las tablas de probabilidad en el caso discreto.

En la práctica, la elección entre construir directamente la distribución conjunta o aplicar eliminación estructurada depende de la escala y la forma del modelo. En redes pequeñas, los métodos directos pueden resultar más eficientes. Sin embargo, en modelos de mayor dimensión, calcular la distribución conjunta completa puede resultar inviable para realizar inferencias precisas, lo que hace necesario recurrir a la eliminación de variables para aprovechar la estructura condicional de la red.

Concluimos así el estudio de las redes bayesianas gaussianas, modelos que permiten inferencia exacta en dominios continuos y cuya estructura computacionalmente eficiente servirá de base para los desarrollos posteriores.

Capítulo 3

SHAP en redes bayesianas gaussianas

En los capítulos anteriores introdujimos el valor de Shapley como una herramienta fundamental para la explicación de modelos predictivos. También presentamos las redes bayesianas gaussianas, una clase de modelos probabilísticos capaces de representar y razonar con distribuciones normales multivariantes de forma estructurada y computacionalmente eficiente.

Este capítulo tiene como objetivo conectar ambas líneas de trabajo mediante el cálculo de valores de Shapley en el contexto de redes bayesianas gaussianas.

3.1. Fundamentos de SHAP

Para ello, nos enfocaremos en uno de los métodos más sólidos y ampliamente utilizados para este fin, **SHAP** [10], una técnica que permite descomponer la predicción de un modelo sobre una instancia concreta en contribuciones individuales asignadas a cada una de sus características. SHAP calcula los valores de Shapley, considerando las características de los datos como jugadores en una coalición y asignando equitativamente la *recompensa* de la predicción entre ellas. Este método proporciona una interpretación detallada tanto de características individuales como de grupos de características, facilitando la comprensión de cómo cada elemento influye en la predicción. Además, SHAP introduce una representación innovadora del valor de Shapley como un método de atribución de características aditivas, lo que nos proporciona explicaciones claras y contrastables al comparar el resultado con la predicción promedio.

Como punto de partida habitual en el aprendizaje automático, dispondremos de un conjunto de entrenamiento $\{y_i, x_i\}_{i=1,\dots,n}$ que se usará para entrenar un modelo predictivo $f(x)$ que aproxime lo mejor posible los valores de y. Si queremos explicar la predicción del modelo $f(x^*)$ para un vector de características específico $x = x^*$, tenemos que la predicción se descompone de la siguiente manera:

$$f(x^*) = \phi_0 + \sum_{j=1}^{k} \phi_j^*$$

donde $\phi_0 = \mathbb{E}[f(x)]$ se puede entender como una predicción que no se debe a ninguna de las características, sino simplemente a la predicción promedio global y ϕ_j^* es el j-ésimo valor de Shapley para la predicción $x = x^*$. El propósito de los valores de Shapley es explicar la diferencia entre la predicción $f(x^*)$ y la predicción promedio global.

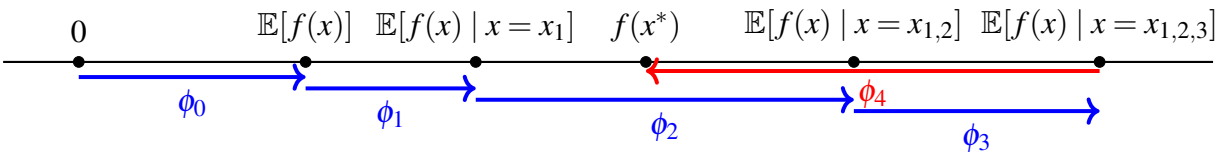


Figura 3.1: Ejemplo de los valores de SHAP para un modelo con cuatro características.

Un modelo de esta forma es un método de atribución de características aditivas que verifica las propiedades enumeradas en la sección 1.1.2.

- **Eficiencia.** La suma de los valores de Shapley para las diferentes características es igual a la diferencia entre la predicción y la predicción promedio global $\sum_{j=1}^{k} \phi_j = f(x^*) - \mathbb{E}[f(x)]$ de ahí, la ecuación anterior. Esta propiedad asegura que la parte del valor de la predicción que no se explica por la predicción media global (ϕ_0), se atribuye y es explicada por las características, de modo que podemos comparar los ϕ_j's de diferentes predicciones $f(x^*)$.
- **Simetría.** Si dos características diferentes contribuyen de manera igual a la predicción, cuando se combinan con cualquier otro subconjunto de características, sus valores de Shapley son iguales sí. No cumplir con este criterio sería inconsistente y daría explicaciones poco confiables.
- **Jugador nulo.** Es natural asignarle un valor de Shapley nulo a una característica que no cambia la predicción, independientemente de con qué otras características se combine.
- **Linealidad.** Cuando una función de predicción consiste en una suma de funciones de predicción, el valor de Shapley para una característica es idéntico a la suma de los valores de Shapley de esa característica de cada una de las funciones de predicción individuales.

Para poder calcular los valores de Shapley en el contexto de la explicación de predicciones, necesitamos definir nuestra función pago $v(S)$ para un cierto subconjunto S; esta debe tratar de asemejarse al valor de $f(x^*)$ cuando solo conocemos el valor de las características del subconjunto S. Para cuantificar esto, usaremos la salida esperada del modelo predictivo, condicionada a los valores de las características x_S, es decir,

$$v(S) = \mathbb{E}_{\Pr}[f(X) \mid X_S = x_S^*]$$

donde Pr es una distribución de probabilidad de referencia. Sin embargo, a pesar de sus virtudes, SHAP sigue presentando importantes limitaciones computacionales. En general, el cálculo

exacto de los valores de Shapley requiere evaluar la función de predicción $f(x)$ del modelo en todas las posibles combinaciones de variables, lo que sugiere un coste exponencial en el número de características.

3.2. Aplicación de SHAP para redes bayesianas gaussianas

3.2.1. Forma cerrada del valor SHAP

No obstante, cuando se cumplen ciertos supuestos estructurales y estadísticos, que resultan naturales en un enfoque probabilístico, es posible calcular estos valores de manera exacta y con complejidad polinómica, lo que hace viable su aplicación en entornos reales de mayor escala.

Concretamente, estudiamos el cálculo de los valores SHAP ϕ_i para funciones con la siguiente forma:

$$F(x) = \mathbb{E}_{P'}[Y \mid \mathbf{X} = \mathbf{x}]$$

Trabajaremos con la esperanza condicional de una variable de salida Y dado un vector de características $\mathbf{X} = (X_1, \ldots, X_n)$, bajo una distribución conjunta normal multivariante P' sobre $(\mathbf{X}, Y)$. Veremos que en este caso es posible obtener una forma cerrada exacta para los valores SHAP, $F(\mathbf{x}) = \mu_Y + w^\top(\mathbf{x} - \mu_\mathbf{X})$, sin necesidad de evaluar el modelo en múltiples coaliciones. Esta solución resultará eficiente si P' está modelada como una red bayesiana gaussiana.

El hecho de haber seleccionado una función de la forma especificada no constituye una simplificación artificial, sino una decisión metodológicamente sólida y justificada:

- Estas funciones surgen de manera natural dentro del ámbito de los modelos probabilísticos estructurados, como las redes bayesianas gaussianas. Al condicionar linealmente cada variable a un subconjunto de otras variables, induce que cualquier variable objetivo Y pueda escribirse como una combinación lineal de sus causas directas y colaterales. Esto permite que la inferencia se reduzca al cálculo de esperanzas condicionales.
- Este enfoque se ajustará particularmente bien al marco de SHAP, donde el objetivo es atribuir la variación entre una predicción y su valor medio a cada característica. El interés se centra en identificar qué aporta cada variable a dicha predicción, basándonos en la estructura de dependencias del modelo probabilístico.
- En la práctica, buena parte de los modelos predictivos se basan en suposiciones de linealidad y ruido gaussiano, como ocurre en la regresión lineal clásica o los modelos gaussianos mixtos. En estos casos, las predicciones no son más que esperanzas condicionales en una distribución normal, lo que refuerza la naturalidad del enfoque adoptado.
- Finalmente, desde el punto de vista computacional, se evita la necesidad de explorar el espacio de subconjuntos, eliminando el coste exponencial de recorrer coaliciones. El cálculo de las expresiones puede realizarse a partir de la matriz de covarianza del modelo

y beneficiarse de la estructura local de una red bayesiana para computarse en tiempo polinómico.

Remarquemos el marco bajo el que vamos a estar trabajando:

- La distribución conjunta P' sobre $(\mathbf{X}, Y)$ es gaussiana multivariante:

$$P' \sim \mathcal{N}(\mu, \Sigma), \quad \text{donde } \mu = \begin{pmatrix} \mu_{\mathbf{X}} \\ \mu_Y \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{\mathbf{XX}} & \Sigma_{\mathbf{X}Y} \\ \Sigma_{Y\mathbf{X}} & \Sigma_{YY} \end{pmatrix}$$

- La función de predicción $F(\mathbf{x})$ es evaluable en tiempo $\mathcal{O}(n)$
- Partiremos de una distribución de referencia completamente factorizada sobre $\mathbf{X}$: $\Pr(x) = \prod_{i=1}^{n} \Pr(x_i)$.

Teorema 3.1. *Sea $F(x) = \mu_Y + w^\top (x - \mu_{\mathbf{X}})$, con $w = \Sigma_{Y\mathbf{X}} \Sigma_{\mathbf{XX}}^{-1}$, y sea $Pr = \prod_i Pr(x_i)$. Entonces, para una instancia $\mathbf{x} = \mathbf{x}^*$, el valor SHAP de la variable i es:*

$$\phi_i = w_i(x_i - \nu_i) \qquad \textit{con } \nu_i = \mathbb{E}_{Pr}[X_i].$$

Demostración: Recordemos que el valor SHAP de una característica i se define como:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \left[v(S \cup \{i\}) - v(S) \right]$$

Tomemos un subconjunto $S \subseteq N \setminus \{i\}$, donde $x_j^{(S)}$ es el vector obtenido al imputar los valores observados en S y completar con las medias marginales de las variables ausentes. Formalmente:

$$x_j^{(S)} = \begin{cases} x_j & \text{si } j \in S, \\ \nu_j & \text{si } j \notin S. \end{cases}$$

Dado que $v(S)$ se define como la esperanza del predictor al fijar las variables en S y completar el resto con la distribución de referencia, esta esperanza coincide, en el caso lineal, con la evaluación de F en el vector imputado $x^{(S)}$, donde las variables fuera de S se reemplazan por sus medias de referencia ν_j. Así, se tiene:

$$v(S) = F(x^{(S)}) = \mu_Y + w^\top (x^{(S)} - \mu_{\mathbf{X}})$$

Aplicando esto también para $S \cup \{i\}$, los vectores imputados $x^{(S)}$ y $x^{(S \cup \{i\})}$ difieren sólo en la coordenada i:

$$x_i^{(S \cup \{i\})} - x_i^{(S)} = x_i - \nu_i, \qquad x_j^{(S \cup \{i\})} - x_j^{(S)} = 0 \quad \forall j \neq i$$

Por lo tanto,

$$v(S \cup \{i\}) - v(S) = w^\top \left(x^{(S \cup \{i\})} - x^{(S)} \right) = w_i(x_i - \nu_i)$$

Como esta diferencia se repite para todas las coaliciones S, llegamos a:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} \cdot w_i(x_i - \nu_i) = w_i(x_i - \nu_i)$$

□

Esta fórmula tiene una interpretación directa y valiosa:

- w_i indica la influencia directa que tiene la variable x_i sobre la predicción y cómo los cambios sobre x_i producen variaciones en $F(\mathbf{x})$.
- $x_i - \nu_i$ mide cuánto se aleja el valor observado de su valor esperado.
- El producto de ambas representa la contribución atribuida a la variable i en la predicción de $F(\mathbf{x}^*)$.

Para calcular $w = \Sigma_{Y\mathbf{X}}\Sigma_{\mathbf{XX}}^{-1}$ es necesario invertir una matriz de dimensión n, lo que conlleva un coste computacional del orden de $\mathcal{O}(n^3)$. Sin embargo, si la distribución P' está representada mediante una red bayesiana gaussiana, este cálculo puede realizarse de forma mucho más eficiente.

3.2.2. El Concepto de Markov blanket

Aunque en principio el cálculo de los valores SHAP podría requerir información sobre la covarianza completa entre todas las variables de entrada, en muchos casos esto es innecesario. Cuando la distribución conjunta $P' = \mathcal{N}(\mu, \Sigma)$ se representa mediante una red bayesiana gaussiana, surge la posibilidad de poder identificar una subestructura que permite reducir la explicación de Y a un subconjunto reducido de variables: su entorno de Markov (o *Markov blanket*).

Definición 18. Sea Y una variable aleatoria en una red bayesiana gaussiana y $\mathbf{X}$ el resto de las variables de la red. El **entorno de Markov** de Y, denotado $\mathrm{MB}(Y)$, es el conjunto mínimo de variables tal que:

$$Y \perp (\mathbf{X} \setminus \mathrm{MB}(Y)) \mid \mathrm{MB}(Y)$$

Es decir, Y es condicionalmente independiente del resto de las variables del modelo dado su entorno de Markov. En nuestro caso para redes dirigidas, este conjunto está compuesto por:

- Los padres de Y.
- Los hijos de Y.
- Los padres de los hijos de Y.

En el contexto gaussiano, esta independencia condicional implica que la distribución condicional $P'(Y \mid \mathbf{X})$ depende únicamente de las variables en $\mathrm{MB}(Y)$. Este hecho nos conduce a los siguientes resultados clave:

- Desde el punto de vista probabilístico, tenemos que $\mathbb{E}_{P'}[Y \mid \mathbf{X} = \mathbf{x}] = \mathbb{E}_{P'}[Y \mid \mathrm{MB}(Y) = \mathbf{x}_{\mathrm{MB}}]$. Por tanto, conocer el valor de $X_i \notin \mathrm{MB}(Y)$ no va a aportar ninguna información adicional para la estimación de Y, una vez conocidas las variables del entorno $\mathrm{MB}(Y)$.
- Desde el punto de vista sobre el algoritmo, el cómputo de $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$ se puede limitar a una inferencia local sobre las variables del entorno de Markov. Gracias al algoritmo de eliminación de variables adaptado para trabajar con la forma canónica, podemos computar esta esperanza mediante sus operaciones básicas aplicadas sobre factores más pequeños.

Para ilustrar de forma clara las ventajas computacionales de nuestro enfoque, conviene destacar que el tamaño del entorno de Markov de la variable objetivo Y, que vamos a denotar como $m := |\mathrm{MB}(Y)|$, no depende directamente del número total de variables n. En la mayoría de los modelos estructurados, se suele cumplir que $m \ll n$, lo que permite desarrollar métodos explicativos cuya complejidad sea sublineal en n.

Antes de adentrarnos en el desarrollo formal de los resultados, introduciremos un ejemplo que nos acompañará a lo largo del capítulo. Este nos permitirá ilustrar, de manera práctica y progresiva, los distintos conceptos y técnicas que iremos presentando.

Ejemplo: Identificación del entorno de Markov

La red que utilizaremos estará compuesta por seis variables gaussianas $\{X_1, X_2, X_3, X_4, X_5, X_6\}$, donde denotaremos como Y a la variable de interés. Consideraremos diferentes elecciones para Y, manteniendo fija la estructura del grafo, con el fin de observar cómo varían los resultados. La estructura causal de nuestra red será la siguiente:

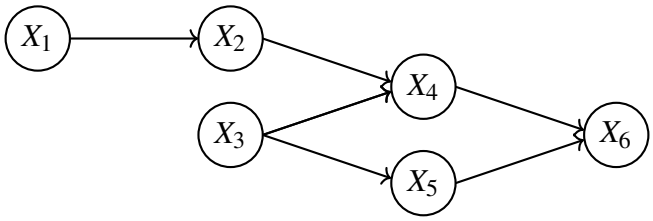


Figura 3.2: Grafo de la red bayesiana gaussiana del ejemplo

En este primer caso, consideramos como variable de interés a $Y = X_2$. Luego su entorno de Markov se obtiene como sigue:

- Padres: $\mathrm{Pa}(X_2) = \{X_1\}$
- Hijos: $\mathrm{Ch}(X_2) = \{X_4\}$
- Co-padres de los hijos: $\mathrm{Pa}(X_4) \setminus \{X_2\} = \{X_3\}$

Por tanto, el entorno de Markov de X_3 está dado por: $\mathrm{MB}(X_2) = \{X_1,\ X_3,\ X_4\}$

Si razonamos a partir de las propiedades de independencia condicional en redes bayesianas gaussianas. se cumple que, una vez conocido el entorno de Markov de una variable, esta se vuelve condicionalmente independiente del resto de las variables:

$$X_2 \perp \{X_5, X_6\} \mid \{X_1, X_3, X_4\} \quad \Rightarrow \quad \mathbb{E}[X_2 \mid X_1, X_3, X_4, X_5, X_6] = \mathbb{E}[X_2 \mid X_1, X_3, X_4]$$

Esto implica que las variables X_5 y X_6 no aportan información adicional sobre X_2 una vez condicionamos sobre su entorno de Markov.

Si exploramos otro caso y elegimos ahora como variable de interés $Y = X_5$, siguiendo el mismo razonamiento que antes, obtenemos que el entorno de Markov de X_5 es: $\mathrm{MB}(X_5) = \{X_3,\ X_4, X_6\}$.

3.2.3. Variables irrelevantes tienen valor SHAP nulo

Una de las consecuencias más destacables del enfoque que estamos siguiendo es que la propia estructura de la red bayesiana gaussiana induce directamente una selección automática de las variables relevantes para la explicación de Y. En particular, veremos que las variables fuera del entorno de Markov de Y tienen un valor de SHAP nulo. Este resultado, además de proporcionar un marco teórico sólido y satisfactorio, también tiene implicaciones prácticas muy relevantes para la eficiencia computacional de nuestro método.

Teorema 3.2. *Sea $P' \sim \mathcal{N}(\mu, \Sigma)$ la distribución conjunta normal multivariante sobre $(\mathbf{X}, Y)$, representada mediante una red bayesiana gaussiana, y sea $MB(Y)$ el entorno de Markov de la variable objetivo Y. Entonces, para cualquier variable $X_i \notin MB(Y)$, su valor SHAP es nulo:*

$$X_i \notin MB(Y) \quad \Longrightarrow \quad \phi_i = 0.$$

Demostración: Recordemos que el valor SHAP de la variable i viene dado por

$$\phi_i \ = \ \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} \Big[v(S \cup \{i\}) - v(S) \Big],$$

donde $v(S)$ denota la esperanza del predictor al fijar las variables de S y marginalizar sobre el resto según la distribución de referencia.

Si $X_i \notin \mathrm{MB}(Y)$, por definición del entorno de Markov en distribuciones gaussianas se cumple:

$$Y \perp X_i \ \mid \ \mathrm{MB}(Y).$$

Esto significa que $v(S \cup \{i\}) = v(S), \ \ \forall S \subseteq N \setminus \{i\}$.

Por tanto, todas las diferencias en la suma de Shapley se anulan, y obtenemos

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} \cdot 0 = 0.$$

□

En resumen, este resultado formaliza que, bajo las hipótesis estructurales y probabilísticas del modelo, las variables que no pertenecen al $\mathrm{MB}(Y)$ no contribuyen a explicar su valor esperado.

Más allá de su interés teórico, esta propiedad tiene un impacto directo en la viabilidad práctica de nuestro enfoque, pues reduce la dependencia de la complejidad computacional respecto al número total de variables. En lugar de ello, el coste del cálculo de los valores SHAP queda determinado principalmente por el tamaño del entorno de Markov de la variable objetivo.

Ejemplo: Valor SHAP nulo para variables irrelevantes

Reutilizamos la red bayesiana gaussiana introducida anteriormente Figura 3.2 y consideramos nuevamente como variable de interés $Y = X_2$. Nuestro objetivo será analizar qué ocurre con el valor SHAP de una variable que no pertenece al entorno de Markov de X_2.

Para ello, asumiremos las siguientes distribuciones condicionales lineales gaussianas para cada nodo de la red:

$$\begin{aligned} X_1 &\sim \mathcal{N}(1,\ 1) \\ X_2 \mid X_1 &\sim \mathcal{N}(2 \cdot X_1 + 1,\ 1) \\ X_3 &\sim \mathcal{N}(0,\ 1) \\ X_4 \mid X_2, X_3 &\sim \mathcal{N}(X_2 + 0{,}5 \cdot X_3,\ 1) \\ X_5 \mid X_3 &\sim \mathcal{N}(X_3,\ 1) \\ X_6 \mid X_4, X_5 &\sim \mathcal{N}(X_4 + X_5,\ 1) \end{aligned}$$

Tal y como ya habíamos adelantado antes el entorno de Markov de X_2 está dado por:

$$\mathrm{MB}(X_2) = \{X_1,\ X_3,\ X_4\}.$$

Procedemos a construir la distribución conjunta inducida por estas ecuaciones, que es una normal multivariante:

$$p(X_1, X_2, X_3, X_4, X_5, X_6) = \mathcal{N}\left(\mu = \begin{pmatrix} 1 \\ 3 \\ 0 \\ 3 \\ 0 \\ 3 \end{pmatrix},\ \Sigma = \begin{pmatrix} 1 & 2 & 0 & 2 & 0 & 2 \\ 2 & 5 & 0 & 5 & 0 & 5 \\ 0 & 0 & 1 & 0.5 & 1 & 1.5 \\ 2 & 5 & 0.5 & 6.25 & 0.5 & 6.75 \\ 0 & 0 & 1 & 0.5 & 2 & 2.5 \\ 2 & 5 & 1.5 & 6.75 & 2.5 & 10.25 \end{pmatrix}\right)$$

El predictor explicativo se basa en la esperanza condicional:

$$F(x) = \mathbb{E}[X_2 \mid X = x].$$

Para distribuciones gaussianas, este predictor adopta la forma lineal:

$$F(x) = \mu_2 + w^\top (x - \mu),$$

donde el vector de coeficientes w viene dado por:

$$w = \Sigma_{X_2,-2}\,\Sigma_{-2,-2}^{-1}.$$

En nuestro caso, para $Y = X_2$, tenemos:

$$\Sigma_{X_2,-2} = (2,\ 0,\ 5,\ 0,\ 5),$$

correspondiente a las covarianzas de X_2 con el resto de variables $\{X_1, X_3, X_4, X_5, X_6\}$. Calculando $\Sigma_{-2,-2}^{-1}$ y multiplicando, se obtiene:

$$w = \big(\underbrace{1.00}_{X_1},\ \underbrace{-0.25}_{X_3},\ \underbrace{0.5}_{X_4},\ \underbrace{0}_{X_5},\ \underbrace{0}_{X_6}\big).$$

Obsérvese que los coeficientes asociados a X_5 y X_6 son exactamente cero. Esto significa que, en la expresión lineal que define $F(x)$, dichas variables no influyen en la predicción:

$$F(x) = \mu_2 + (x_1 - \mu_1) - \frac{1}{4}\cdot(x_3 - \mu_3) + \frac{1}{2}\cdot(x_4 - \mu_4).$$

Esto tiene consecuencias directas sobre los valores SHAP. Dado que:

$$F(x^{(S\cup\{i\})}) - F(x^{(S)}) = w_i(x_i - \mu_i),$$

se cumple que si $w_i = 0$, entonces todos los términos de la suma que define ϕ_i son nulos:

$$\phi_i = \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(n-|S|-1)!}{n!}\cdot 0 = 0.$$

Por tanto, para la variable X_5, que no pertenece al entorno de Markov de X_2, tenemos:

$$\boxed{\phi_5 = 0}.$$

Este resultado muestra el contenido del Teorema 3.2, podemos ver como las variables fuera del entorno de Markov de Y reciben un coeficiente nulo en el predictor lineal, lo que implica automáticamente que sus valores SHAP son cero.

3.2.4. Cálculo eficiente de SHAP en RGB

Recordemos el resultado del Teorema 3.1, donde para $F(x) = \mathbb{E}_{P'}[Y \mid \mathbf{X} = \mathbf{x}]$ y Pr de referencia factorizada,los valores SHAP de la instancia x^* verifican

$$\phi_i = w_i\big(x_i^* - \nu_i\big), \quad \text{donde} \quad w = \Sigma_{Y\mathbf{X}}\,\Sigma_{\mathbf{XX}}^{-1}.$$

Por tanto, el problema computacional se reduce a obtener el vector de coeficientes w. La inversión de $\Sigma_{\mathbf{XX}}$ cuesta $\mathcal{O}(n^3)$, pero en una red bayesiana gaussiana podemos evitarlo explotando la forma canónica y la eliminación de variables.

Identidad clave en forma canónica y estrategia computacional

Sea $(\mathbf{X}, Y)$ un vector aleatorio gaussiano y la forma canónica de la distribución conjunta escrita como:

$$p(\mathbf{x}, y) \propto \exp\Big(-\tfrac{1}{2} \begin{bmatrix} \mathbf{x} \\ y \end{bmatrix}^\top \begin{bmatrix} J_{\mathbf{XX}} & J_{\mathbf{X}Y} \\ J_{Y\mathbf{X}} & J_{YY} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ y \end{bmatrix} + \begin{bmatrix} h_{\mathbf{X}} \\ h_Y \end{bmatrix}^\top \begin{bmatrix} \mathbf{x} \\ y \end{bmatrix} - g \Big)$$

donde $J = \Sigma^{-1}$ es la matriz de precisión y h el vector canónico asociado.

Al condicionar en $\mathbf{X} = \mathbf{x}$ se obtiene una normal canónica en Y con parámetros:

$$p(y \mid \mathbf{x}) \propto \exp\Big(-\tfrac{1}{2} y^\top J_{YY} y + (h_Y - J_{Y\mathbf{X}}\mathbf{x})^\top y \Big)$$

Para una densidad proporcional a $\exp\big(-\frac{1}{2} y^\top A y + b^\top y \big)$ la esperanza es $A^{-1}b$. Aplicando esto con $A = J_{YY}$ y $b = h_Y - J_{YX}\mathbf{x}$ obtenemos inmediatamente:

$$\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}] \;=\; J_{YY}^{-1}\big(h_Y - J_{Y\mathbf{X}}\mathbf{x}\big).$$

Dado que $J = \Sigma^{-1}$ y $h = J\mu$, si particionamos correctamente las identidades se tiene:

$$h_Y = J_{Y\mathbf{X}}\,\mu_{\mathbf{X}} + J_{YY}\,\mu_Y \quad \Rightarrow \quad J_{YY}^{-1} h_Y = J_{YY}^{-1} J_{Y\mathbf{X}}\,\mu_{\mathbf{X}} + \mu_Y$$

y sustituyendo en la expresión anterior:

$$\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}] = J_{YY}^{-1} h_Y - J_{YY}^{-1} J_{Y\mathbf{X}}\mathbf{x} = \mu_Y + J_{YY}^{-1} J_{Y\mathbf{X}}\,\mu_{\mathbf{X}} - J_{YY}^{-1} J_{Y\mathbf{X}}\mathbf{x} = \mu_Y - J_{YY}^{-1} J_{Y\mathbf{X}}\,(\mathbf{x} - \mu_{\mathbf{X}}).$$

Comparando con la forma lineal $F(\mathbf{x}) = \mu_Y + w^\top(\mathbf{x} - \mu_{\mathbf{X}})$ obtenemos la identidad clave que usaremos:

$$\boxed{w \;=\; -J_{Y\mathbf{X}} J_{YY}^{-1}}$$

Vamos a buscar calcular w sin invertir Σ completa. Para ello, basta con obtener los bloques J_{YY} y $J_{Y\mathbf{X}}$ de la matriz de precisión y estos en una RBG gaussiana se pueden construir mediante la suma de contribuciones locales canónicas de cada CPD (*Conditional Probability Distribution*).

Regla detallada para un CPD lineal–gaussiano: derivación y justificación

Sea Z un nodo con padres $\mathrm{Pa}(Z) = \{P_1, \ldots, P_r\}$, modelado mediante una distribución condicional lineal-gaussiana:

$$Z \mid \mathrm{Pa}(Z) \;\sim\; \mathcal{N}\big(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z),\; \sigma_Z^2\big),$$

donde $\beta_Z \in \mathbb{R}^r$ son los coeficientes de regresión, α_Z es el intercepto y σ_Z^2 la varianza residual.

Nuestro objetivo es escribir el factor local $p(Z \mid \mathrm{Pa}(Z))$ en *forma canónica* y derivar explícitamente los bloques de $J^{(Z)}$, el vector $h^{(Z)}$ y la constante $g^{(Z)}$.

Expansión del logaritmo de la densidad condicional

Trabajaremos con el logaritmo de la densidad condicional porque la representación canónica gaussiana se obtiene directamente de su forma cuadrática en el logaritmo. La matriz de precisión J coincide con la hessiana de $-\log p(Z \mid \mathrm{Pa}(Z))$. Así, escribimos:

$$\log\left(p(Z \mid \mathrm{Pa}(Z))\right) = -\frac{1}{2\sigma_Z^2}\left(Z - \alpha_Z - \beta_Z^\top \mathrm{Pa}(Z)\right)^2 + \mathrm{cte}.$$

Expandimos el cuadrado:

$$\begin{aligned}-\frac{1}{2\sigma_Z^2}\left(Z - \alpha_Z - \beta_Z^\top \mathrm{Pa}(Z)\right)^2 &= -\frac{1}{2\sigma_Z^2}\left(Z^2 - 2Z(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z)) + (\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z))^2\right) \\ &= -\frac{1}{2\sigma_Z^2}Z^2 + \frac{1}{\sigma_Z^2}Z(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z)) - \frac{1}{2\sigma_Z^2}(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z))^2.\end{aligned}$$

Desarrollando el término $(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z))^2$ se obtiene:

$$(\alpha_Z + \beta_Z^\top \mathrm{Pa}(Z))^2 = \alpha_Z^2 + 2\alpha_Z \beta_Z^\top \mathrm{Pa}(Z) + (\beta_Z^\top \mathrm{Pa}(Z))^2.$$

Identificación de los bloques canónicos

La forma canónica de un factor sobre $(\mathrm{Pa}(Z), Z)$ es

$$-\tfrac{1}{2}\begin{bmatrix}\mathrm{Pa}(Z)\\ Z\end{bmatrix}^\top \begin{bmatrix}J_{\mathrm{Pa}(Z)\mathrm{Pa}(Z)} & J_{\mathrm{Pa}(Z)Z}\\ J_{Z\mathrm{Pa}(Z)} & J_{ZZ}\end{bmatrix}\begin{bmatrix}\mathrm{Pa}(Z)\\ Z\end{bmatrix} + \begin{bmatrix}h_{\mathrm{Pa}(Z)}\\ h_Z\end{bmatrix}^\top \begin{bmatrix}\mathrm{Pa}(Z)\\ Z\end{bmatrix} - g.$$

Ahora compararemos con la expansión anterior y identificamos según el grado en $\mathrm{Pa}(Z)$ y Z.

- **Término cuadrático en Z.** El coeficiente de Z^2 es $-\frac{1}{2\sigma_Z^2}$ y al compararlo con $-\frac{1}{2}J_{ZZ}Z^2$ deducimos
$$J_{ZZ}^{(Z)} \;=\; \sigma_Z^{-2}.$$

- **Término mixto $Z\cdot\mathrm{Pa}(Z)$.** Su coeficiente es $\frac{1}{\sigma_Z^2}Z\beta_Z^\top \mathrm{Pa}(Z)$ y se corresponde con el término de la forma canónica $-\mathrm{Pa}(Z)^\top J_{\mathrm{Pa}(Z)Z}Z$:
$$-\mathrm{Pa}(Z)^\top J_{\mathrm{Pa}(Z)Z}^{(Z)}Z \;=\; \frac{1}{\sigma_Z^2}Z\beta_Z^\top \mathrm{Pa}(Z) \quad\Longrightarrow\quad J_{\mathrm{Pa}(Z)Z}^{(Z)} \;=\; -\,\sigma_Z^{-2}\beta_Z.$$
Observar que $J_{Z\mathrm{Pa}(Z)}^{(Z)} = (J_{\mathrm{Pa}(Z)Z}^{(Z)})^\top$.

- **Término cuadrático en** $\mathrm{Pa}(Z)$**.** Aparece como $-\frac{1}{2\sigma_Z^2}(\beta_Z^\top \mathrm{Pa}(Z))^2$, que tras desarollar y comparar:

$$-\frac{1}{2\sigma_Z^2}\mathrm{Pa}(Z)^\top(\beta_Z\beta_Z^\top)\mathrm{Pa}(Z) = -\tfrac{1}{2}\,\mathrm{Pa}(Z)^\top\big(\sigma_Z^{-2}\beta_Z\beta_Z^\top\big)\mathrm{Pa}(Z) \quad \to \quad J^{(Z)}_{\mathrm{Pa}(Z)\mathrm{Pa}(Z)} = \sigma_Z^{-2}\beta_Z\beta_Z^\top$$

- **Términos lineales (vector** h**).**
  - Del término $\frac{1}{\sigma_Z^2} Z\,\alpha_Z$ se obtiene la componente lineal en Z:

$$h_Z^{(Z)} \;=\; \sigma_Z^{-2}\,\alpha_Z.$$

  - Del término $-\frac{1}{2\sigma_Z^2}\cdot 2\alpha_Z\beta_Z^\top \mathrm{Pa}(Z) = -\frac{\alpha_Z}{\sigma_Z^2}\beta_Z^\top \mathrm{Pa}(Z)$ se obtiene la componente lineal en $\mathrm{Pa}(Z)$:

$$h_{\mathrm{Pa}}(Z)^{(Z)} \;=\; \sigma_Z^{-2}\,\alpha_Z\,\beta_Z.$$

- **El término constante** $g^{(Z)}$**).** Se toma como $g^{(Z)} = \frac{1}{2}\,\sigma_Z^{-2}\,\alpha_Z^2 \;+\; \frac{1}{2}\log(2\pi\sigma_Z^2)$, notar que este va acompañado de una constante si se desea seguir la normalización.

En aplicaciones prácticas, basta con los bloques de la matriz de precisión J, aunque hayamos incluido también los términos lineales (h) y la constante (g).

Resumiendo, la contribución local canónica del factor $p(Z \mid \mathrm{Pa}(Z))$ sobre las variables $(\mathrm{Pa}(Z), Z)$ viene dada por los bloques y vectores:

$J^{(Z)}_{ZZ} = \sigma_Z^{-2}$	$J^{(Z)}_{\mathrm{Pa}(Z)Z} = -\,\sigma_Z^{-2}\,\beta_Z$	$J^{(Z)}_{\mathrm{Pa}(Z)\mathrm{Pa}(Z)} = \sigma_Z^{-2}\,\beta_Z\beta_Z^\top$
$h_Z^{(Z)} = \sigma_Z^{-2}\,\alpha_Z$	$h^{(Z)}_{\mathrm{Pa}(Z)} = \sigma_Z^{-2}\,\alpha_Z\,\beta_Z$	$g^{(Z)} = \frac{1}{2}\sigma_Z^{-2}\alpha_Z^2 + cte$

Composición global como suma de contribuciones locales

En una red bayesiana gaussiana, la distribución conjunta se factoriza como producto de factores locales:

$$p(x_1,\ldots,x_n) \;=\; \prod_Z p\big(Z \mid \mathrm{Pa}(Z)\big) \qquad p^{(Z)}(x) \propto \exp\big(-\tfrac{1}{2}x^\top J^{(Z)}x + h^{(Z)\top}x - g^{(Z)}\big)$$

Al multiplicar factores en el dominio exponencial, los exponentes *se suman*:

$$\prod_Z p^{(Z)}(x) \propto \exp\Big(-\tfrac{1}{2}x^\top\Big(\sum_Z J^{(Z)}\Big)x + \Big(\sum_Z h^{(Z)}\Big)^\top x - \sum_Z g^{(Z)}\Big).$$

Por tanto, los **parámetros globales** (J,h,g) de la distribución conjunta en forma canónica son:

$$J \;=\; \sum_Z J^{(Z)}, \qquad h \;=\; \sum_Z h^{(Z)}, \qquad g \;=\; \sum_Z g^{(Z)}.$$

Cada CPD lineal–gaussiano induce únicamente un bloque local de un tamaño restringido a la variable Z y sus padres. Al sumar todas estas contribuciones se obtiene la matriz de precisión global J y el vector canónico h. Esta representación en forma canónica es exacta y ventajosa, pues

- Permite construir J simplemente acumulando bloques locales, sin necesidad de invertir la matriz de covarianzas global.
- Facilita aplicar marginalizaciones y eliminaciones mediante complementos de Schur.
- Hace posible implementar algoritmos eficientes usando estructuras dispersas por bloques.

En consecuencia, la regla general establecida constituye la base computacional para el cálculo de los valores SHAP en RBG, pues reduce el problema a operaciones polinómicas sobre factores pequeños.

Algoritmo 2: Cálculo de w a partir de CPDs locales (RBG)

Entrada: RBG con variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, objetivo Y
Salida: vector w tal que $F(x) = \mu_Y + w^\top (x - \mu_{\mathbf{X}})$

Inicializar:

$$\varphi_X \leftarrow 0 \quad \forall X \in \mathrm{MB}(Y)$$

Definir:

$$\varphi_Y \leftarrow \sigma_Y^{-2}$$

for cada hijo $X \in \mathrm{Ch}(Y)$ **do** ▷ $Ch(Y)$ es el conjunto de hijos de Y
    $\varphi_X \mathrel{-}= \sigma_X^{-2}\,\beta_{XY}$ ▷ Actualizar contribución directa
    for cada padre $Z \in \mathrm{Pa}(X)$ **do**
        $\varphi_Z \mathrel{+}= \sigma_X^{-2}\,\beta_{XY}\,\beta_{XZ}$ ▷ Propagar contribución hacia Z
    end for
end for
for cada padre $Z \in \mathrm{Pa}(Y)$ **do**
    $\varphi_Z \mathrel{-}= \sigma_Y^{-2}\,\beta_{YZ}$ ▷ Ajustar contribución de Z
end for
return $\{-\frac{\varphi_X}{\varphi_Y} \;:\; \forall X \in \mathrm{MB}(Y)\}$ ▷ Devuelve el vector w

---

Con este procedimiento se evita la construcción de bloques canónicos globales, trabajando únicamente con los parámetros locales de las CPDs de los nodos en $\mathrm{MB}(Y)$.

Análisis de complejidad del procedimiento propuesto

Introducimos primero los parámetros relevantes que determinan el coste:

- c_Y: número de hijos de la variable objetivo Y.

- d_Y: número de padres de la variable objetivo Y.
- $d_{\mathrm{Ch}(Y)}$: máximo número de padres entre los hijos de Y.

El algoritmo consta de tres fases principales:

1. Inicialización de los acumuladores φ para Y y las variables en $\mathrm{MB}(Y)$. Este paso requiere un tiempo lineal en el tamaño del entorno de Markov y su coste es despreciable dentro del algoritmo.
2. Recorrr los hijos de Y y cada hijo $X \in \mathrm{Ch}(Y)$ induce una actualización sobre sí mismo y sobre cada uno de sus padres. Como cada X tiene a lo sumo $d_{\mathrm{Ch}(Y)}$ padres, este paso tiene coste
$$\mathcal{O}\big(c_Y\, d_{\mathrm{Ch}(Y)}\big)$$
3. Recorrer de los padres de Y, actualizando su acumulador correspondiente requiere
$$\mathcal{O}(d_Y)$$

El cálculo final de los coeficientes

$$w_X = -\frac{\varphi_X}{\varphi_Y}, \qquad X \in \mathrm{MB}(Y)$$

es inmediato y no afecta a la complejidad.

Coste total. El procedimiento completo tiene por tanto complejidad

$$\boxed{\mathcal{O}\big(c_Y\, d_{\mathrm{Ch}(Y)} + d_Y\big)}$$

Este coste depende únicamente de la estructura local en torno a la variable objetivo Y, y no del número total n de nodos de la red. Además, se cumple que

$$c_Y d_{\mathrm{Ch}(Y)} + d_Y \;\geq\; |\mathrm{MB}(Y)|$$

pues siempre hay que recorrer al menos tantos nodos como contiene dicho entorno. Considerando el peor casos, donde los padres de cada hijo son todos distintos, tendríamos que

$$c_Y d_{\mathrm{Ch}(Y)} + d_Y \;\leq\; \mathcal{O}(|\mathrm{MB}(Y)|^2)$$

En consecuencia, el coste del procedimiento como máximo cuadrático en el tamaño del entorno de Markov, lo que garantiza su eficiencia al depender únicamente de la estructura local de la red.

Ejemplo: Cálculo de w mediante eliminación local en una RBG

Consideremos nuevamente la red bayesiana gaussiana introducida en la Figura 3.2, con las mismas distribuciones condicionales y fijamos la variable objetivo $Y = X_2$. Nuestro objetivo es aplicar el algoritmo anteior para calcular el vector w en la representación:

$$F(x) = \mu_2 + w^\top (x - \mu),$$

Ejecución numérica sobre el ejemplo. Destacar que con las CPDs con las que trabajamos $\sigma^2 = 1$, por tanto $\sigma^{-2} = 1$.

$$X_2 \mid X_1 :\ \beta_{2,1} = 2 \quad (\text{padre de } Y \text{ es } X_1),$$
$$X_4 \mid X_2, X_3 :\ \beta_{4,2} = 1,\ \beta_{4,3} = 0.5 \quad (\text{hijo } X_4 \in \mathrm{Ch}(Y)).$$

El entorno de Markov ya conocido es $\mathrm{MB}(X_2) = \{X_1,\ X_3,\ X_4\}$.

1. Inicialización:
$$\varphi_{X_1} = 0, \quad \varphi_{X_3} = 0, \quad \varphi_{X_4} = 0, \qquad \varphi_Y = \sigma_Y^{-2} = 1$$

2. Procesamos los hijos de Y. Aquí $\mathrm{Ch}(Y) = \{X_4\}$. Para X_4:
$$\varphi_{X_4} \leftarrow \varphi_{X_4} - \sigma_4^{-2}\beta_{4,2} = 0 - 1 \cdot 1 = -1$$

   Ahora actualizamos las entradas para los padres de X_4, es decir, $Z \in \{Y, X_3\}$:

$$\varphi_Y \leftarrow \varphi_Y + 1 \cdot \beta_{4,2}\beta_{4,2} = 1 + 1 \cdot 1 \cdot 1 = 2$$
$$\varphi_{X_3} \leftarrow \varphi_{X_3} + 1 \cdot \beta_{4,2}\beta_{4,3} = 0 + 1 \cdot 1 \cdot 0.5 = 0.5$$

3. Procesamos los padres de Y. Aquí $\mathrm{Pa}(Y) = \{X_1\}$ y $\beta_{2,1} = 2$:
$$\varphi_{X_1} \leftarrow \varphi_{X_1} - \sigma_Y^{-2}\beta_{2,1} = 0 - 1 \cdot 2 = -2.$$

Cálculo final de w. Aplicando la fórmula de salida:

$$w_X \ = \ -\frac{\varphi_X}{\varphi_Y} \quad \text{para } X \in \{X_1, X_3, X_4\},$$

Finalmente:

$$\boxed{w = (w_{X_1}, w_{X_3}, w_{X_4}) = (1,\ -0.25,\ 0.5).}$$

En particular, como $w_{X_5} = w_{X_6} = 0$ se tiene $\phi_5 = \phi_6 = 0$ con independencia de x^*, que es la conclusión que buscábamos ilustrar: las variables fuera del Markov blanket no contribuyen al score SHAP en nuestro modelo.

Capítulo 4

Ejemplo práctico

El propósito de este capítulo es ilustrar de manera práctica los resultados teóricos y algorítmicos desarrollados en el capítulo anterior. Para ello, vamos a realizar un estudio utilizando el siguiente conjunto de datos, *Wine Quality Dataset*, bastante conocido en el ámbito del aprendizaje automático. Este contiene medidas físico-químicas de vinos tintos y blancos, junto con una variable de calidad que ha sido determinada por expertos en una escala entera de 0 a 10.

Nuestro objetivo va a ser entrenar redes bayesianas gaussianas para ambos tipos de vino, tanto tinto como blanco. Para poder aplicar nuestro enfoque, utilizaremos la variable *quality* como objetivo, y posteriormente aplicaremos el procedimiento de cálculo eficiente de valores SHAP presentado. Asimismo, para tratar de apoyar la validez de las cotas de complejidad deducidas teóricamente, complementaremos el análisis experimentando con datos sintéticos.

4.1. Análisis con datos reales

Preparación de los datos

El conjunto de datos de calidad del vino está compuesto por dos subconjuntos:

- **Vino tinto**: 1599 observaciones con 11 variables predictoras y la variable objetivo *quality*.
- **Vino blanco**: 4898 observaciones con la misma estructura de variables.

Las variables predictoras recogen medidas químicas y físico-químicas: acidez fija (*fixed acidity*), acidez volátil (*volatile acidity*), ácido cítrico (*citric acid*), azúcar residual (*residual sugar*), cloruros (*chlorides*), dióxido de azufre libre (*free sulfur dioxide*), dióxido de azufre total (*total sulfur dioxide*), densidad (*density*), pH, sulfatos (*sulphates*) y alcohol (*alcohol*). En ambos casos, la variable *quality* toma valores enteros entre 3 y 8 en el vino tinto, y entre 3 y 9 en el vino blanco. Las siguientes tablas con las frecuencias reflejan una cierta escasez de ejemplos en los extremos, especialmente para valores bajos.

Calidad	Frecuencia
3	10
4	53
5	681
6	638
7	199
8	18

Cuadro 4.1: Distribución en vino tinto

Calidad	Frecuencia
3	20
4	163
5	1457
6	2198
7	880
8	175
9	5

Cuadro 4.2: Distribución en vino blanco

Dataset	# Instancias	# Variables predictoras	Rango de calidad
Vino tinto	1599	11	3 – 8
Vino blanco	4898	11	3 – 9

Cuadro 4.3: Resumen general de los datasets de vino tinto y vino blanco.

Resumen descriptivo de los datos

A continuación, presentaremos un resumen numérico de las variables para ambos tipos de vino que utilizaremos en los experimentos. Las tablas recogerán las estadísticas básicas habituales y servirán de referencia para detectar las diferencias de nuestros datasets.

Tabla de estadísticos: vino tinto

Variable	N	Media	Desv.	Mín	25 %	50 %	75 %	Máx
fixed acidity	1599	8.320	1.741	4.600	7.100	7.900	9.200	15.900
volatile acidity	1599	0.528	0.179	0.120	0.390	0.520	0.640	1.580
citric acid	1599	0.271	0.195	0.000	0.090	0.260	0.420	1.000
residual sugar	1599	2.539	1.410	0.900	1.900	2.200	2.600	15.500
chlorides	1599	0.087	0.047	0.012	0.070	0.079	0.090	0.611
free sulfur dioxide	1599	15.875	10.460	1.000	7.000	14.000	21.000	72.000
total sulfur dioxide	1599	46.468	32.895	6.000	22.000	38.000	62.000	289.000
density	1599	0.997	0.002	0.990	0.996	0.997	0.998	1.004
pH	1599	3.311	0.154	2.740	3.210	3.310	3.400	4.010
sulphates	1599	0.658	0.170	0.330	0.550	0.620	0.730	2.000
alcohol	1599	10.423	1.066	8.400	9.500	10.200	11.100	14.900
quality	1599	5.636	0.808	3.000	5.000	6.000	6.000	8.000

Cuadro 4.4: Estadísticos descriptivos — vino tinto (red wine).

Tabla de estadísticos: vino blanco

Variable	N	Media	Desv.	Mín	25 %	50 %	75 %	Máx
fixed acidity	4898	6.855	0.844	3.800	6.300	6.800	7.300	14.200
volatile acidity	4898	0.278	0.101	0.080	0.210	0.260	0.320	1.100
citric acid	4898	0.334	0.121	0.000	0.270	0.320	0.390	1.660
residual sugar	4898	6.391	5.072	0.600	1.700	5.200	9.900	65.800
chlorides	4898	0.046	0.022	0.009	0.036	0.043	0.050	0.346
free sulfur dioxide	4898	35.308	17.007	2.000	23.000	34.000	46.000	289.000
total sulfur dioxide	4898	138.361	42.498	9.000	108.000	134.000	167.000	440.000
density	4898	0.994	0.003	0.987	0.992	0.994	0.996	1.039
pH	4898	3.188	0.151	2.720	3.090	3.180	3.280	3.820
sulphates	4898	0.490	0.114	0.220	0.410	0.470	0.550	1.080
alcohol	4898	10.514	1.231	8.000	9.500	10.400	11.400	14.200
quality	4898	5.878	0.886	3.000	5.000	6.000	6.000	9.000

Cuadro 4.5: Estadísticos descriptivos — vino blanco (white wine).

Vino tinto

- La variable *residual sugar* presenta una media baja ($\approx$2.54) junto con una dispersión moderada, lo que indica que la mayoría de los vinos tintos del conjunto tienen poco azúcar residual.
- Los valores de *alcohol*, *density* y *pH* muestran una notable estabilidad, evidenciada por sus bajas desviaciones típicas y la consistencia en sus medias y cuartiles.
- Las variables *free sulfur dioxide* y *total sulfur dioxide* presentan amplitud considerable, lo que sugiere la presencia de valores extremos.

Vino blanco

- La variable *residual sugar* es notablemente mayor en media ($\approx$6.39) y dispersión, distinguiendo claramente los vinos blancos del tinto.
- Las medidas de *density*, *pH* y *alcohol* muestran un comportamiento similar y con unos valores parecidos a los observados para el vino tinto.
- Las concentraciones de *total sulfur dioxide* y *free sulfur dioxide* también muestran una amplia variabilidad, aunque estos toman valores medios y máximos más altos que aquellos registrados en los vinos tintos.

Con estos puntos claros, procederemos a aprender la estructura de las RBG para cada subconjunto, a estimar los CPD lineales-gausianos y a aplicar nuestro procedimiento eficiente para calcular los valores SHAP.

Aprendizaje de la estructura de la Red Bayesiana Gaussiana

La implementación del aprendizaje de la red bayesiana gaussiana (RGB) se realiza en Python mediante la librería `pgmpy`, utilizando el algoritmo *Hill Climb Search* con puntuación BIC y restringiendo el número máximo de padres por nodo. Esta limitación afecta directamente la complejidad de la red y el tamaño del entorno de Markov, influyendo tanto la interpretabilidad como el coste computacional.

Aplicando el procedimiento descrito y fijando un máximo de 4 padres por nodo, se obtienen las siguientes estructuras de redes bayesianas gaussianas, como se muestra a continuación.

Variable	Descripción	Color
X_1	fixed_acidity	
X_2	citric_acid	
X_3	density	
X_4	volatile_acidity	
X_5	residual_sugar	
X_6	quality	
X_7	sulphates	
X_8	alcohol	
X_9	chlorides	
X_{10}	pH	
X_{11}	free_sulfur_dioxide	
X_{12}	total_sulfur_dioxide	

Cuadro 4.6: Correspondencia entre variables originales, nodos del grafo y color asignado.

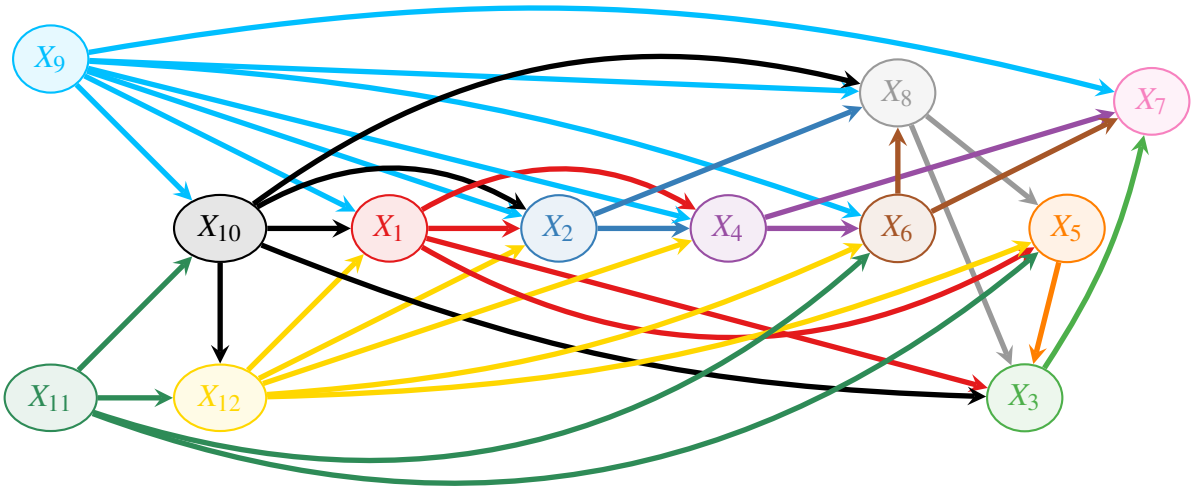


Figura 4.1: Grafo de la estructura de la RBG aprendida para el vino tinto.

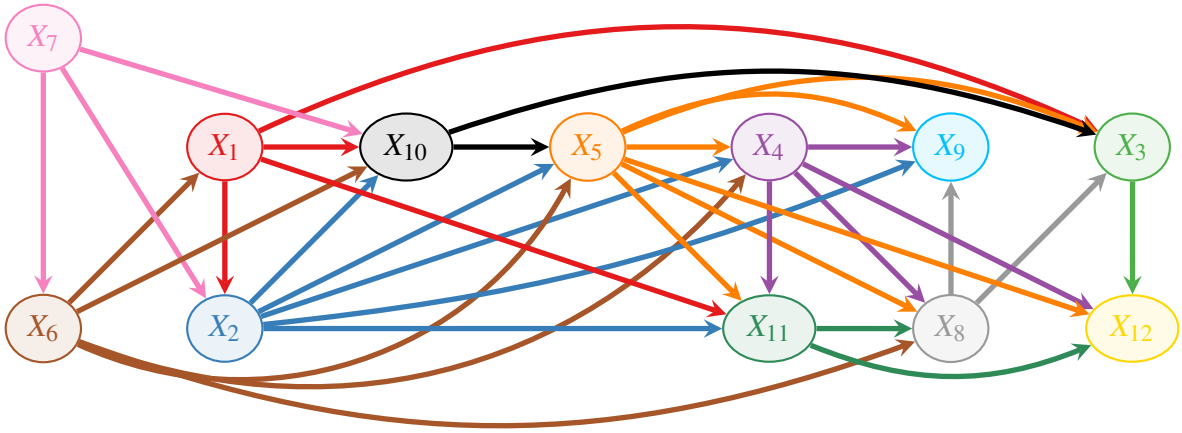


Figura 4.2: Grafo de la estructura de la RBG aprendida para el vino blanco.

Si comparamos ambas estructuras, se aprecian diferencias notables. En el vino tinto, la calidad depende principalmente de factores de carácter negativo, como la *volatile_acidity* y los *chlorides*, mientras que en el vino blanco la dependencia directa recae sobre los *sulphates*, lo que refleja la diversidad de procesos enológicos que determinan la calidad. Otro aspecto destacable es la mayor presencia de la variable *residual_sugar* en la red del vino blanco, donde además aparece como hija de *quality*, evidenciando así el característico dulzor que posee el vino blanco frente al tinto.

Esto manifiesta la necesidad de tratar por separado ambos modelos, puesto que la red codifica relaciones estadísticas específicas en cada caso, lo que afecta a los valores SHAP que vayamos a obtener.

Valores SHAP en las redes de vino tinto y blanco

Una vez aprendidas las estructuras de las RBG para ambos subconjuntos de datos, aplicamos el procedimiento que hemos desarrollado para calcular eficientemente los valores SHAP para la variable objetivo (*quality*). Para asegurarnos de poder realizar comparaciones, ambas redes fueron entrenadas con la misma configuración. La siguiente figura presenta los resultados obtenidos con un máximo de 4 padres. En el Anexo II cubrimos otros casos.

En la Figura 4.3 observamos una representación en forma de barras, donde para cada variable aparecen dos contribuciones: en color rojo se muestran los valores SHAP correspondientes al vino tinto, mientras que en gris se representan los del vino blanco. Aquellas variables que no pertenecen al entorno de Markov de la variable objetivo se indican con una cruz del color correspondiente, ya que sus valores SHAP son nulos.

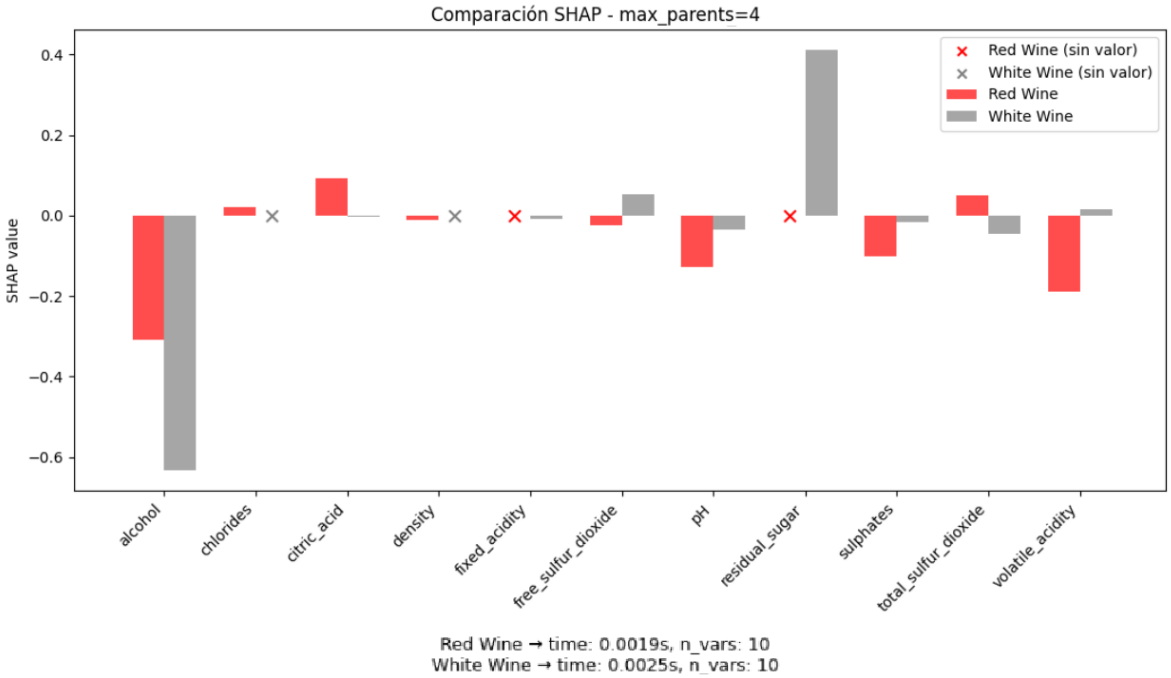


Figura 4.3: Comparación de valores SHAP para RBG bajo la misma configuración para vino tinto y vino blanco.

En el caso del vino tinto, los resultados refuerzan lo observado en la estructura de la red. Las variables con mayor influencia sobre la calidad son *volatile_acidity* y *alcohol*, ambas con valores SHAP negativos. A estas se suman *pH* y *sulphates*, que también tienen magnitud negativa. En conjunto, la calidad del vino tinto aparece fuertemente condicionada por factores de carácter adverso, lo que coincide con el comportamiento enológico esperado.

Por su parte, los valores SHAP del vino blanco resultan en general más bajos en términos absolutos. Salvo dos variables que destacan, *alcohol*, con una aportación negativa clara, y *residual_sugar*, con influencia positiva. Este último resultado es especialmente coherente con lo comentado anteriormente, apoyando el dulzor del vino blanco como un reflejo de su calidad.

En ambos vinos se observa que el *alcohol* influye de manera negativa en la calidad, mostrando que este podría ser factor limitante. Al mismo tiempo, las diferencias detectadas entre tinto y blanco dejan claro que el tipo de vino es un aspecto determinante en la forma en que las variables enológicas inciden sobre la valoración final.

4.2. Validación con datos sintéticos

Procedimiento

Además de trabajar con los conjuntos de datos reales de vino tinto y vino blanco, resulta conveniente considerar un escenario sintético que podamos controlar. El objetivo en este caso

no es realizar un análisis de nuestro conjunto de datos, sino validar el procedimiento propuesto bajo distintas configuraciones estructurales de la RBG.

El uso de datos sintéticos presenta dos ventajas principales:

- Permite controlar el número total de variables n y el tamaño esperado del entorno de Markov de la variable objetivo, variando parámetros como el número máximo de padres por nodo .
- Generar un marco donde la estructura de la red y los parámetros son conocidos, facilitando comprobar que el algoritmo implementado respeta las propiedades teóricas.

De esta manera, los experimentos sintéticos sirven como una verificación práctica de la eficiencia computacional del método, más allá de la interpretación concreta sobre un conjunto de datos.

Para la generación de redes sintéticas, utilizamos una función que construye un conjunto de nodos $\{X_0, X_1, \ldots, X_{n-1}\}$ de tamaño n, junto con una estructura acíclica dirigida que respeta la restricción en el número de padres por variable. La función asigna distribuciones condicionales lineales gaussianas con parámetros generados aleatoriamente a partir de distribuciones normales y uniformes. Finalmente, genera un conjunto de datos de tamaño N mediante muestreo sobre el DAG, de forma que los valores de cada variable se obtienen a partir de sus CPDs y de los valores previamente generados de sus padres.

En este contexto, se fija una variable objetivo, nosotros tomaremos X_0. Para después utilizar el conjunto de datos generado para estudiar el comportamiento del cálculo de los valores SHAP y del coste computacional del procedimiento.

Experimento de complejidad

Con el fin de validar empíricamente el análisis teórico, realizaremos un conjunto de experimentos sobre redes bayesianas gaussianas sintéticas de distinto tamaño.

Procederemos de la siguiente manera:

- Trabajaremos con redes con distinto número de nodos $n \in \{15, 20, 25, \ldots, 75, 80\}$.
- Para cada valor de n, se modifica el número máximo de padres permitido por nodo *max_parents* $\in \{3, 5, 7, 9, 11\}$, lo que afecta directamente al tamaño esperado del entorno de Markov.
- Para cada par $(n, \textit{max\_parents})$ se generaron datos sintéticos fijando como variable objetivo X_0.
- A cada red se le aplica nuestro procedimiento de cálculo de SHAP, registrando el tiempo de cómputo y el tamaño del entorno de Markov $\mathrm{MB}(X_0)$.
- Se reportan los valores medios y desviaciones estándar del tiempo de cómputo y del tamaño del entorno de Markov para cada configuración.

Análisis

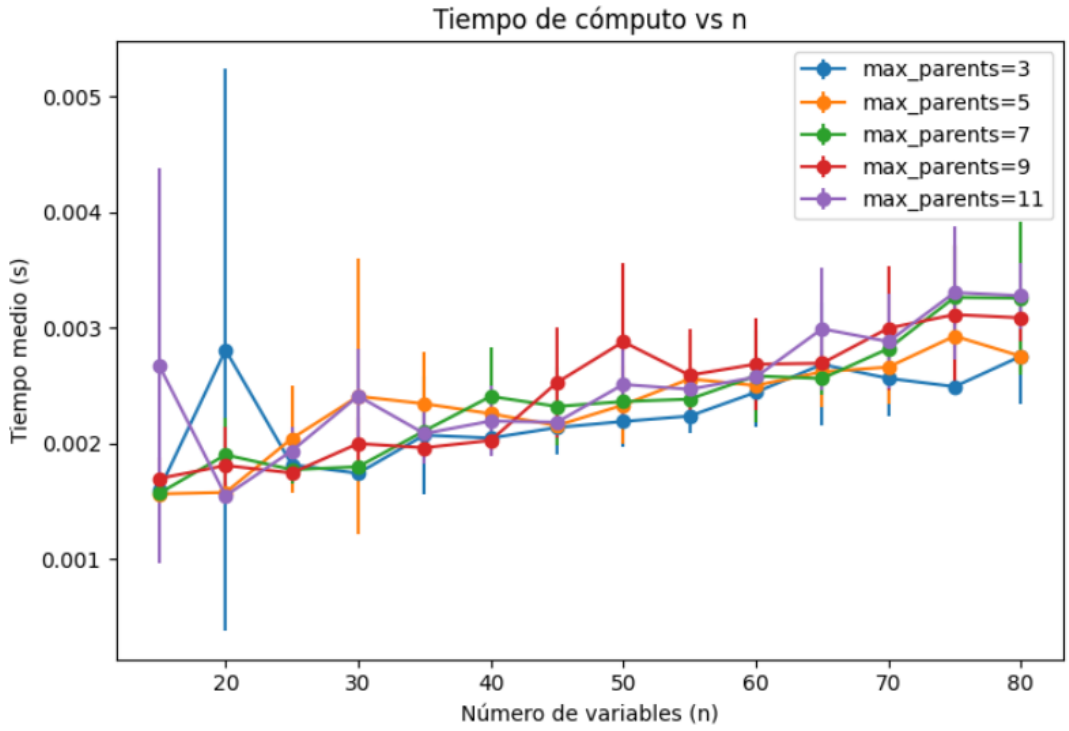


Figura 4.4: Tiempo medio de cómputo en función del número de variables n.

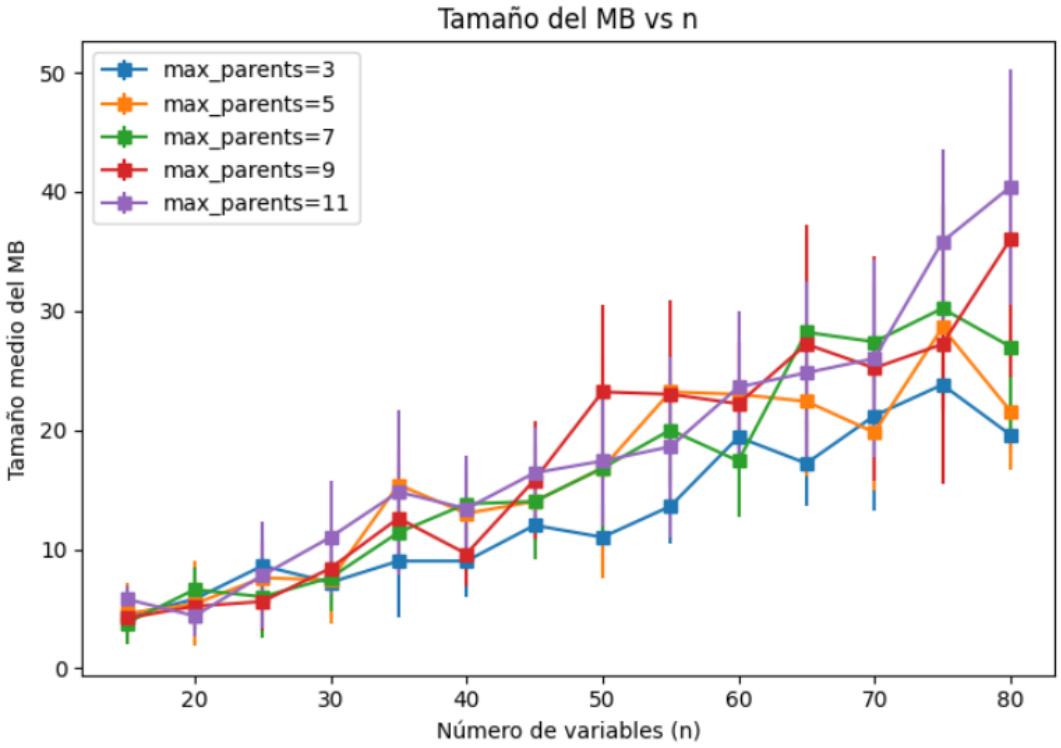


Figura 4.5: Tamaño medio del entorno de Markov en función de n.

Los resultados del experimento se presentan en la Figura 4.4 y la Figura 4.5. Estas gráficas muestran la evolución del tiempo de cómputo y del tamaño del entorno de Markov en función del número de nodos n y del parámetro *max_parents*.

Observamos que el tiempo medio de cómputo se mantiene en el orden de milisegundos (0,0016 s–0,0033 s) incluso al aumentar el número de nodos de 15 a 80 y el máximo de padres de 3 a 11. Esta variación es moderada y sugiere una tendencia de crecimiento prácticamente lineal en n. Para confirmar que los resultados siguen manifestando esta alta eficiencia del método frente al crecimiento de la red, sería necesario extender el análisis a redes con un número de nodos considerablemente mayor.

Conclusión

En resumen, este experimento con datos sintéticos confirma la eficiencia práctica del algoritmo propuesto. Los resultados observados concuerdan con la parte teórica y muestran que el coste computacional permanece reducido incluso en redes de mayor tamaño. No obstante, convendría extender este análisis a escenarios de mayor escala, con un número más amplio de repeticiones y un rango más elevado tanto en el tamaño de la red como en el parámetro *max_parents*, lo que permitiría validar el comportamiento del método en redes mucho más grandes. En cualquier caso, los resultados actuales refuerzan la aplicabilidad del procedimiento en redes de tamaño considerable, donde los enfoques globales habituales resultarían inviables.

Bibliografía

[1] AAS, K.; JULLUM, M. Y LØLAND, A., *Explaining Individual Predictions When Features Are Dependent: More Accurate Approximations to Shapley Values*, 2019, https://arxiv.org/pdf/1903.10464

[2] ALI, S.; ABUHMED, T.; EL-SAPPAGH, S.; MUHAMMAD, K.; ALONSO, J.; CONFALONIERI, R.; GUIDOTTI, R.; DEL SER, J.; DÍAZ-RODRÍGUEZ, N. Y HERRERA, F., *Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence*, 2023, https://www.researchgate.net/publication/370111593_Explainable_Artificial_Intelligence_XAI_What_we_know_and_what_is_left_to_attain_Trustworthy_Artificial_Intelligence

[3] BARCELO, P.; COMINETTI, R. Y MORGADO, M., *When is the Computation of a Feature Attribution Method Tractable?* , 2025, https://arxiv.org/pdf/2501.02356

[4] BARREDO ARRIETA, A.; DÍAZ-RODRÍGUEZ, N.; DEL SER, J.; BENNETOT, A.; TABIK, S.; BARBADO, A.; GARCIA, S.; GIL-LOPEZ, S.; MOLINA, D.; BENJAMINS, R.; CHATILA, R. Y HERRERA, F., *Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI*, 2020, https://doi.org/10.1016/j.inffus.2019.12.012.

[5] COMISIÓN EUROPEA, *Proyecto de ley de inteligencia artificial*, Legislación de la UE en progreso, 2024, https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf Consultado a fecha de 07/06/2024

[6] DEVOS, M. Y KENT, D. A., *Game Theory A Playful Introduction*, American Mathematical Society, Volumen 80, 2016, https://bookstore.ams.org/stml-80

[7] HASTIE, T.; TIBSHIRANI, R. Y FRIEDMAN, J., *The Elements of Statistical Learning*, Data Mining, Inference, and Prediction, Segunda edición, 2009, https://hastie.su.domains/Papers/ESLII.pdf

[8] KOCHENDERFER, M. J.; WHEELER, T. A. Y WRAY, K. H., *Algorithms for Decision Making*, 2022, https://algorithmsbook.com/decisionmaking/

[9] KOLLER, D. Y FRIEDMAN, N., *Probabilistic Graphical Models: Principles and Techniques*, 2009, http://mcb111.org/w06/KollerFriedman.pdf

[10] MOLNAR, C. *Interpretable machine learning. A Guide for Making Black Box Models Explainable*, 2019, https://christophm.github.io/interpretable-ml-book/

[11] PÉREZ, J.; JIMENO, J. L. Y CERDÁ, E., *Teoría de Juegos*, Pearson-Prentice Hall, Madrid, 2004, https://elvisjgblog.wordpress.com/wp-content/uploads/2018/02/teorc3ada-de-juegos-joaquc3adn-pc3a9rez-2004.pdf

[12] SHAPLEY, L. S., *A Value for n-Person Games*, Contributions to the Theory of Games II, Princeton University Press, 1953, https://www.rand.org/content/dam/rand/pubs/papers/2021/P295.pdf

[13] VAN DEN BROECK, G.; LYKOV, A.; SCHLEICH, M. Y SUCIU, D., *On the Tractability of SHAP Explanations*, 2020, https://arxiv.org/pdf/2009.08634

Anexos

Apéndice I

Contenido adicional de redes bayesianas

I.1. Ejemplo discreto de una red bayesiana

Para motivar el uso de redes bayesianas, vamos a pensar que disponemos de una colección de n variables binarias. En principio, una distribución conjunta sobre este conjunto requeriría $2^n - 1$ parámetros independientes. Sin embargo, bajo el pretexto de asumir la independencia de las variables, bastaría únicamente con n parámetros. Por otro lado, es cierto que estamos realizando una hipótesis un tanto potente y poco realista, pero nos da pie a ver cómo ciertas estructuras de independencia permiten representaciones mucho más compactas.

Veamos ahora un ejemplo sencillo que ilustra esta idea. Supongamos que una empresa desea contratar estudiantes que se acaban de graduar y busca identificar a los candidatos que podríamos considerar inteligentes. Sin embargo, la inteligencia no es directamente observable, por lo que la empresa recurre a indicadores indirectos.

Vamos a empezar con un ejemplo sencillo con dos variables aleatorias binarias:

- I, que representa el nivel de inteligencia del candidato (alto o bajo).
- E, que representa su puntuación en un examen (alta o baja).

La forma directa de modelar esta situación es especificar la distribución conjunta $P(I,S)$ con una tabla de probabilidades que consta de cuatro posibles combinaciones de valores:

I	i^0	i^0	i^1	i^1
E	e^0	e^1	e^0	e^1
$P(I,E)$	0.665	0.035	0.060	0.240

Esta tabla describe completamente la distribución conjunta, pero no refleja ninguna posible relación estructural entre las variables. Este detalle es importante, pues parece natural suponer que la inteligencia de un candidato va a influir en su puntuación del Examen.

Esta dependencia puede aprovecharse para factorizar la distribución utilizando la regla de la cadena, $P(I,E) = P(I) \cdot P(E \mid I)$. Bajo esta representación, especificamos por separado la distribución marginal de I y la distribución condicional de E dado I:

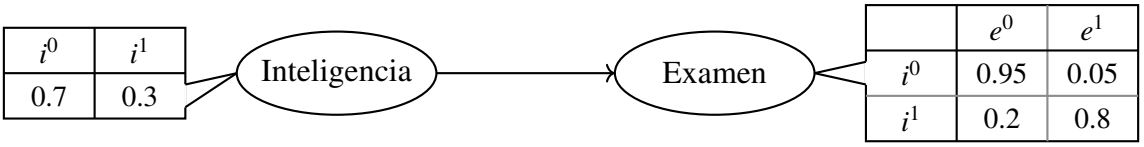


Figura I.1: Estructura condicionada para el Examen dada la Inteligencia

Esta representación condicionada se mantiene coherente con nuestra intuición del problema. En términos de parametrización, tanto la representación conjunta como esta factorizada requieren tres parámetros independientes. No obstante, la segunda forma introduce una estructura que será esencial cuando se incrementa el número de variables y que nos ayudará a reducir drásticamente el número de parámetros necesarios si se aprovechan adecuadamente las dependencias entre variables.

Mientras que en el ejemplo anterior solo teníamos una relación de independencia marginal total entre variables, ahora introducimos una nueva dimensión: la calificación académica (N). Esta tomará tres valores posibles: A, B y C, denotados respectivamente como n_1, n_2 y n_3 . De modo que el espacio probabilístico completo está ahora definido sobre tres variables: I, E y N, lo que implica una distribución conjunta con $2 \times 2 \times 3 = 12$ entradas, y por tanto 11 parámetros independientes en el caso general. En este nuevo caso, ninguna de las variables es independiente en términos marginales, lo cual lo convierte en un caso más realista y útil. Intuitivamente, volveremos a tener que la inteligencia (I) afecta tanto a las notas (N) como al examen (E), pero además existe una aparente dependencia entre N y E, pues es natural esperar que un estudiante con buenas Notas sea más dado a tener un buen Examen. Aquí es donde entra la suposición de independencia condicional y la hipótesis de que el Examen y las Notas solo están relacionados a través de la Inteligencia y que, una vez sabemos información sobre esta, podemos considerar que son independientes. No obstante, estamos otra vez apoyándonos en algo que puede no ser completamente verdad, pero gracias a ello vamos a poder construir nuevamente una representación compacta de la distribución conjunta.

Mediante un razonamiento probabilístico sencillo, si asumimos la independencia condicional, llegamos a la siguiente descomposición del modelo conjunto:

$$P(I,E,N) = P(I) \cdot P(E \mid I) \cdot P(N \mid I)$$

Para especificar completamente esta distribución, bastará con conocer las siguientes distribuciones:

- La marginal $P(I)$.
- Las condicionales $P(E \mid I)$ y $P(N \mid I)$.

La representación factorizada obtenida resulta ser más eficiente. En lugar de requerir 11 parámetros independientes, esta parametrización solo necesita siete: uno para $P(I)$, dos para $P(E \mid I)$ y cuatro para $P(N \mid I)$.

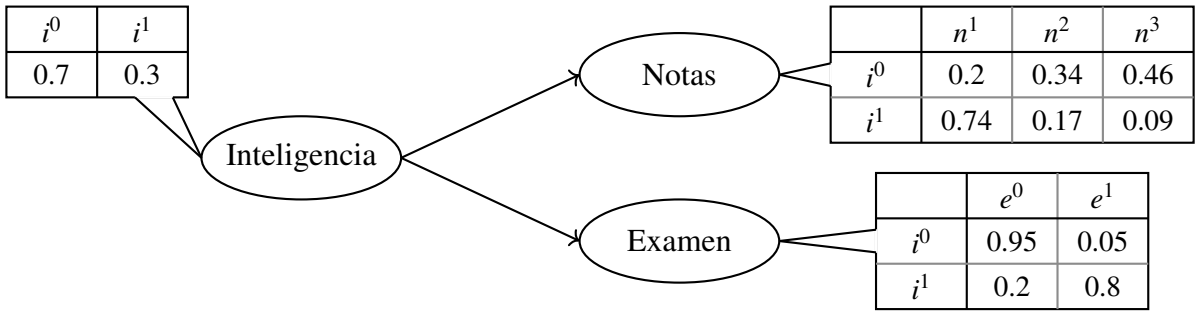


Figura I.2: Red bayesiana simple con independencia condicional

Además, otra ventaja que nos ofrece este tipo de representación es la posibilidad de incorporar una nueva variable al modelo sin la necesidad de rehacer por completo toda la distribución.

Este tipo de factorizaciones, basadas en supuestos de independencia condicional, no solo permiten una representación más compacta, sino también una mayor flexibilidad en la construcción y extensión del modelo. Una generalización directa de este enfoque es el modelo Naive Bayes.

El modelo de Naive Bayes parte del supuesto de que todas las características $X_1, \ldots, X_n$ son condicionalmente independientes entre sí, una vez observado su antecesor común, llamado habitualmente clase y que denotaremos por C.

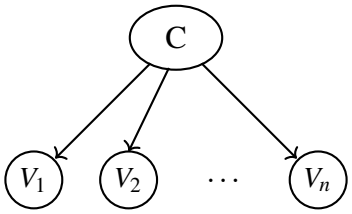


Figura I.3: Esquema general del modelo Naive Bayes

Tenemos una estructura con un nodo central C con una única arista dirigida a cada uno de sus nodos hijos X_i, teniendo que para todo i, $(X_i \perp X_{-i} \mid C)$. A pesar de la simplicidad de este esquema, este grafo codifica una distribución conjunta no trivial:

$$P(C, X_1, \ldots, X_n) = P(C) \prod_{i=1}^{n} P(X_i \mid C)$$

Esta forma factorizada ofrece una gran ventaja computacional al permitir poder representar distribuciones de alta dimensión con un número de parámetros que crece linealmente con el número de variables observadas.

No obstante, esta eficiencia viene a costa de una suposición que, en muchos casos, resulta poco realista. Las variables observables suelen estar correlacionadas entre sí incluso cuando

se condiciona por la clase. Esto puede limitar la capacidad del modelo para capturar ciertas dependencias estructurales presentes en los datos. En la práctica, esta limitación se manifiesta en predicciones menos precisas cuando la independencia condicional no se cumple de forma aproximada.

A pesar de ello, el modelo de Naive Bayes funciona sorprendentemente bien en muchos contextos reales, en particular cuando se dispone de pocos datos o se requiere un modelo interpretable y de bajo coste computacional. Sin embargo, cuando se desea modelar relaciones más complejas entre las variables, resulta necesario relajar la hipótesis de independencia e introducir estructuras más generales. Esto nos lleva de forma natural a las redes bayesianas, que permiten representar explícitamente las dependencias condicionales entre variables y modelar de forma más rica las interacciones en un sistema.

Para ilustrar esto, retomamos el ejemplo del estudiante, pero lo ampliamos incorporando nuevas variables que pueden influir en la evaluación de un candidato. Al modelo que ya teníamos antes le añadiremos dos nuevas variables:

- D, que representa la dificultad del curso que ha hecho el estudiante (fácil o difícil). Esta afectará directamente a las notas del candidato.
- C, que representa la calidad de la carta de recomendación escrita por el profesor (fuerte o débil). Esta solo va a depender de la nota que ha conseguido el estudiante.

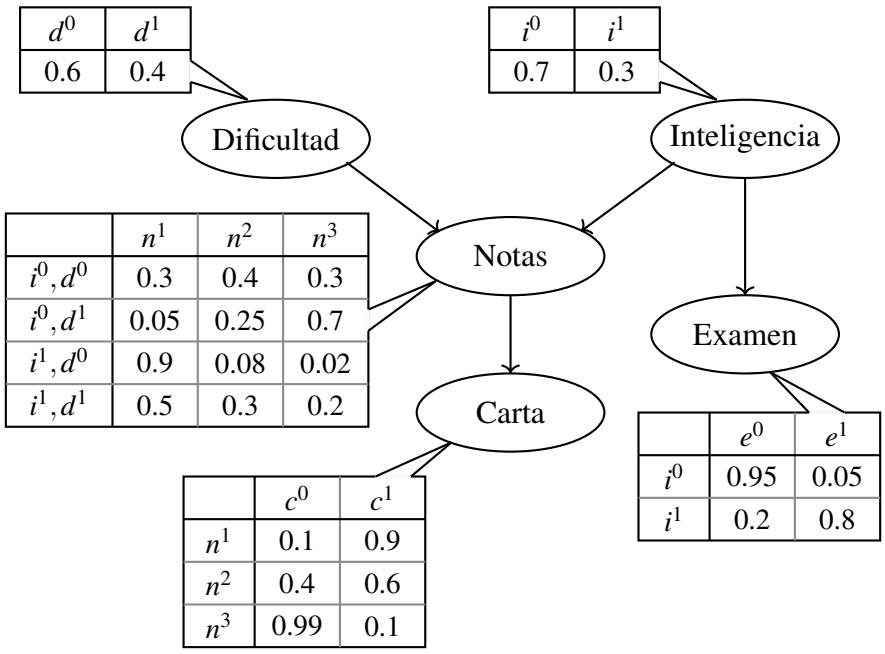


Figura I.4: Esquema de la red bayesiana del ejemplo del estudiante

Si quisiéramos modelar la distribución conjunta $P(I,D,N,E,C)$ sin ninguna estructura, necesitaríamos especificar 47 variables. Sin embargo, gracias a la estructura dirigida de una red bayesiana, podemos factorizar esta distribución en función de las dependencias locales entre

las variables. Para ello, asignamos a cada variable una distribución de probabilidad condicional que describe cómo varía en función de sus padres.

Estas distribuciones locales pueden representarse con tablas, que a través de sus relaciones nos permiten calcular la probabilidad conjunta de cualquier combinación de valores para nuestras variables.

Este enfoque, que generaliza la idea de factorización del modelo Naive Bayes, se formaliza en lo que conocemos como la regla de la cadena para redes bayesianas. En nuestro caso, la factorización completa de la distribución conjunta es:

$$P(I,D,N,E,C) = P(I) \cdot P(D) \cdot P(N \mid I,D) \cdot P(E \mid I) \cdot P(C \mid N)$$

Una de las principales virtudes de las redes bayesianas, más allá de su eficiencia al representar distribuciones conjuntas complejas, es que nos permiten razonar bajo incertidumbre de forma estructurada. En términos generales, al observar el valor de una variable, modificamos las probabilidades que asignamos a otras, según cómo estén conectadas en la red. Este fenómeno es lo que entendemos como razonamiento probabilístico, y puede adoptar distintas formas dependiendo de la dirección de la inferencia y la naturaleza de las conexiones.

Cuando utilizamos el conocimiento de una causa para predecir un efecto, hablamos de razonamiento causal. Por ejemplo, si sabemos que un estudiante es inteligente, podemos estimar de forma más certera si es probable que obtenga buenas calificaciones. Este tipo de razonamiento fluye en la misma dirección que las flechas del grafo, es decir, desde los nodos padres hacia sus hijos. Por otro lado, cuando observamos un efecto y buscamos inferir una posible causa, realizamos razonamiento evidencial. Si vemos que un estudiante obtuvo una mala nota, podríamos reconsiderar nuestras creencias sobre su nivel de inteligencia. Este tipo de razonamiento fluye en contra de la dirección de las flechas del grafo.

Una situación más sutil ocurre cuando dos o más causas comparten un efecto común. En estos casos, la información sobre una de las causas puede alterar nuestra creencia sobre la otra. En general, si sabemos que un efecto ha ocurrido, y luego aprendemos que una de sus posibles causas está presente, esto disminuye la probabilidad de que otra causa alternativa también lo esté. Es decir, las causas compiten por explicar el efecto. Por ejemplo, si observamos un mal resultado y sabemos que el curso era difícil, eso puede atenuar nuestra sospecha de que el estudiante no era inteligente.

El impacto de una observación sobre otras variables depende directamente de la estructura de la red. Aunque no todas las variables están conectadas entre sí, la información puede propagarse por caminos indirectos definidos por las dependencias condicionales. Esto significa que una variable puede influir en otra de forma directa o a través de múltiples rutas. A veces, estos efectos se refuerzan; otras veces, se neutralizan. Lo importante es que las redes bayesianas permiten modelar y calcular estas interacciones de manera precisa. Obtener nueva evidencia no solo ajusta una creencia aislada, sino que puede alterar todo el sistema de probabilidades,

afectando variables aparentemente lejanas al propagar sus efectos a través de la red.

Este modelo nos brinda una manera formal de razonar sobre qué información importa, cuándo importa y por qué. Esa capacidad de razonamiento no es arbitraria, sino que está determinada por la propia arquitectura del modelo. En este esquema, cada nodo depende directamente solo de sus padres, y esa relación estructural es la que marcará qué información es relevante. Sin embargo, esta regla no implica que, al conocer los padres de una variable, esta se vuelva automáticamente independiente del resto. En algunos casos, saber información adicional sobre variables descendientes puede afectar, mostrando que la influencia en la red puede surgir por caminos que no son directos.

I.2. Ejemplo discreto del algoritmo de eliminación de variables

Vamos a volver a proceder usando de nuevo una versión ampliada de una red bayesiana basada en el ejemplo del estudiante. La distribución conjunta de las variables involucradas está dada por la siguiente factorización:

$$\begin{aligned} P(Co,D,I,N,E,C,T,S) &= P(Co)P(D \mid Co)P(I)P(N \mid I,D)P(E \mid I)P(C \mid N)P(T \mid C,E)P(S \mid N,T) \\ &= \phi_{Co}(Co)\phi_D(D,Co)\phi_I(I)\phi_N(N,I,D)\phi_E(E,I)\phi_C(C,N)\phi_T(T,C,E)\phi_S(S,N,T) \end{aligned}$$

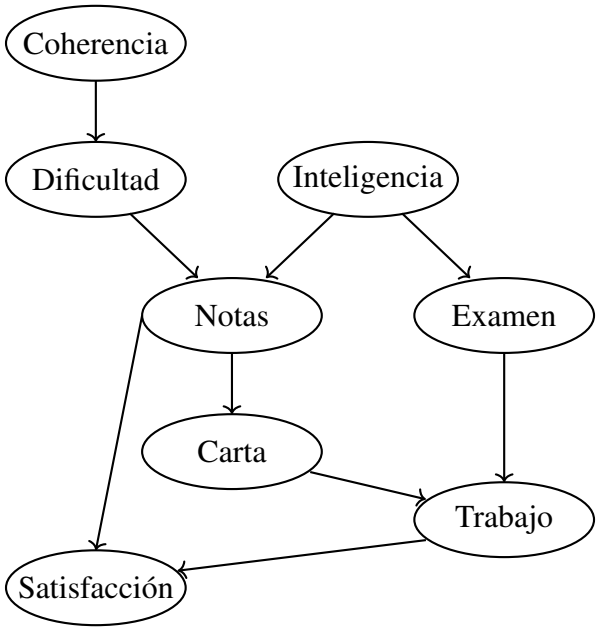


Figura I.5: Esquema de la red bayesiana del ejemplo del estudiante ampliada

El objetivo que plantearemos será la aplicación del algoritmo de eliminación de variables para computar $P(T)$. En otras palabras, vamos a calcular la distribución marginal de la variable T, eliminando el resto de las variables en el siguiente orden Co,D,I,S,N,E,C.

Veamos en detalle este procedimiento:

1. **Eliminamos C**. Los factores involucrados son $\phi_C(C)$ y $\phi_D(D,C)$. Se agrupan y se marginaliza C.

$$\psi_1(C0,D) = \phi_{Co}(Co) \cdot \phi_D(D,Co)$$
$$\tau_1(D) = \sum_{Co} \psi_1(Co,D)$$
Factores: $\tau_1(D),\ \phi_I(I),\ \phi_N(N,I,D),\ \phi_E(E,I),\ \phi_C(C,N),\ \phi_T(T,C,E),\ \phi_S(S,N,T)$

2. **Eliminamos D**. Repetiremos lo mismo aplicandolo a el nuevo factor $\tau_1(D)$ junto con $\phi_N(N,I,D)$.

$$\psi_2(N,I,D) = \phi_N(N,I,D) \cdot \tau_1(D)$$
$$\tau_2(N,I) = \sum_D \psi_2(N,I,D)$$
Factores: $\tau_2(N,I),\ \phi_I(I),\ \phi_E(E,I),\ \phi_C(C,N),\ \phi_T(T,C,E),\ \phi_S(S,N,T)$

3. **Eliminamos I**. Los factores que nos importan ahora son $\tau_2(N,I),\ \phi_I(I)$ y $\phi_E(E,I)$.

$$\psi_3(N,I,E) = \phi_I(I) \cdot \phi_E(E,I) \cdot \tau_2(N,I)$$
$$\tau_3(N,E) = \sum_I \psi_3(N,I,E)$$
Factores: $\tau_3(N,E),\ \phi_C(C,N),\ \phi_T(T,C,E),\ \phi_S(S,N,T)$

4. **Eliminamos S**. Solo interviene el factor $\phi_S(S,N,T)$, ya que es el único con S.

$$\tau_4(N,T) = \sum_S \phi_S(S,N,T)$$
Factores: $\tau_4(N,T),\ \tau_3(N,E),\ \phi_C(C,N),\ \phi_T(T,C,E)$

5. **Eliminamos N**. Trabajaremos como hasta ahora con $\tau_4(N,T),\ \tau_3(N,E)$ y $\phi_C(C,N)$.

$$\psi_5(N,T,C,E) = \tau_4(N,T) \cdot \tau_3(N,E) \cdot \phi_C(C,N)$$
$$\tau_5(T,C,E) = \sum_N \psi_5(N,T,C,E)$$
Factores: $\tau_5(T,C,E),\ \phi_T(T,C,E)$

6. **Eliminamos E y C**. Solo nos quedan dos factores, $\tau_5(T,C,E)$ y $\phi_T(T,C,E)$, que dependen de las mismas variables. Agrupamos estos factores para después marginalizar primero en E y luego en C.

$$\psi_6(T,C,E) = \tau_5(T,C,E) \cdot \phi_T(T,C,E)$$
$$\tau_6(T,C) = \sum_E \psi_6(T,C,E)$$
$$\tau_7(T) = \sum_C \tau_6(T,C)$$

Este factor $\tau_7(T)$ representa la distribución marginal $P(T)$, que era el objetivo de nuestra consulta.

Cabe destacar que el orden que hemos tomado en la eliminación de variables nos ha permitido avanzar de forma eficiente. Sin embargo, este no es el único posible. Si siguiéramos otro orden alternativo, el resultado final sería el mismo, pero el algoritmo podría haber generado factores intermedios de mayor tamaño, lo que incrementaría significativamente el coste computacional del proceso de inferencia.

I.3. La inferencia en redes bayesianas discretas es NP-completo

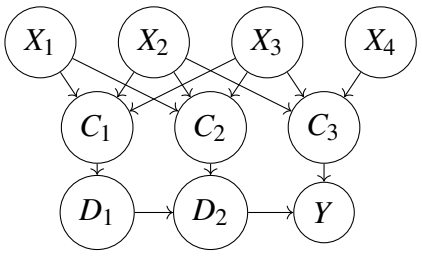
Desde un punto de vista teórico, se ha demostrado que el problema de la inferencia en redes bayesianas es NP-completo en su versión restringida a redes con variables binarias. Esto implica que, en principio, no existe un algoritmo eficiente que resuelva el problema en todos los casos posibles. Una forma habitual para demostrarlo es mediante una reducción desde el problema 3-SAT, que es un problema canónico NP-completo. Esta técnica permite mostrar que resolver problemas de inferencia en redes bayesianas es, al menos, tan difícil como resolver una instancia arbitraria de 3-SAT. Consideremos la siguiente fórmula 3-SAT:

$$F(x_1,x_2,x_3,x_4) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

Aquí, $\vee, \wedge$ y $\neg$ representan la disyunción, conjunción y negación lógicas, respectivamente. Esta fórmula es una conjunción de cláusulas, donde cada cláusula es una disyunción de tres literales.

A partir de esta fórmula se puede construir una red bayesiana que codifique su estructura lógica. En dicha red:

- Cada variable x_i se representa como un nodo X_i, cuya distribución de probabilidad es uniforme.
- Cada cláusula C_j se representa mediante un nodo C_j que depende como padres de las variables que aparecen en ella.
- La distribución condicional de cada nodo cláusula se define de forma determinista, asignando probabilidad 1 a las asignaciones que satisfacen la cláusula y 0 a aquellas que no lo hacen.



Red bayesiana del problema de 3-SAT

Para representar la satisfacibilidad global de la fórmula, se introduce un nodo adicional Y. Este solamente valdrá 1 si y solo si todas las cláusulas se satisfacen simultáneamente, es decir, la probabilidad $P(Y = 1)$ es estrictamente positiva si y solo si existe al menos una asignación de valores a las variables $x_1, \ldots, x_4$ que satisfaga toda la fórmula. Por tanto, decidir si $P(Y = 1) > 0$ es equivalente a resolver el problema de satisfacibilidad de la fórmula 3SAT original.

Por tanto, la inferencia exacta requiere tiempo exponencial en el tamaño de la red, y no es razonable esperar una solución eficiente que funcione para todos los casos posibles. No obstante, esta dificultad teórica no impide que en muchas aplicaciones prácticas se logre realizar inferencia de forma eficiente, gracias a propiedades estructurales de las redes o al uso de métodos de inferencia aproximada.

Apéndice II

Código utilizado

```
# Paso 1: Cargar y preparar Wine Quality (red y white)
import pandas as pd
from IPython.display import display

# URLs UCI
red_url   = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
white_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"

# 1) Descargar / cargar (sep=';')
try:
    red_wine   = pd.read_csv(red_url, sep=';')
    white_wine = pd.read_csv(white_url, sep=';')
    print("Descarga: OK")
except Exception as e:
    raise RuntimeError("Error descargando los ficheros. Si estas en Colab asegurate de tener conexion a Internet.") from e

# 2) Incorporar una columna 'type' para distinguir (temporalmente)
red_wine['type']   = 'red'
white_wine['type'] = 'white'

# (opcional) vista rapida combinada para verificar
print("\nMuestras rapidas (red top 3, white top 3):")
display(pd.concat([red_wine.head(3), white_wine.head(3)], ignore_index=True))

# 3) Separar datasets y eliminar 'type' (no usado como predictor)
red_data   = red_wine.drop(columns=['type']).copy()
white_data = white_wine.drop(columns=['type']).copy()

```

```python
# 4) Confirmar variable objetivo 'quality' presente
assert 'quality' in red_data.columns, "La variable 'quality' no esta en red_data"
assert 'quality' in white_data.columns, "La variable 'quality' no esta en white_data"
print("\nConfirmacion: 'quality' encontrada en ambos datasets")

# 5) Resumen para la memoria: shape, variables y primeras filas
for name, df in [('Red wine', red_data), ('White wine', white_data)]:
    print("\n" + "="*60)
    print(f"{name} - shape: {df.shape}")
    print("Variables / columnas:")
    print(df.columns.tolist())
    print("\nPrimeras 5 filas:")
    display(df.head())
    print("\nResumen numerico (describe):")
    display(df.describe().T)   # transpuesto para mejor lectura en la memoria
    print("\nDistribucion de 'quality' (conteos):")
    print(df['quality'].value_counts().sort_index().to_string())
    print("="*60)
```

```
Descarga: OK

Muestras rapidas (red top 3, white top 3):
    fixed acidity   volatile acidity   citric acid   residual sugar
0             7.4               0.70          0.00              1.9
1             7.8               0.88          0.00              2.6
2             7.8               0.76          0.04              2.3
3             7.0               0.27          0.36             20.7
4             6.3               0.30          0.34              1.6
5             8.1               0.28          0.40              6.9
    chlorides   free sulfur dioxide   total sulfur dioxide   density
0       0.076                  11.0                   34.0    0.9978
1       0.098                  25.0                   67.0    0.9968
2       0.092                  15.0                   54.0    0.9970
3       0.045                  45.0                  170.0    1.0010
4       0.049                  14.0                  132.0    0.9940
5       0.050                  30.0                   97.0    0.9951
         pH   sulphates   alcohol   quality   type
0      3.51        0.56       9.4         5    red
1      3.20        0.68       9.8         5    red
2      3.26        0.65       9.8         5    red
3      3.00        0.45       8.8         6  white
4      3.30        0.49       9.5         6  white
5      3.26        0.44      10.1         6  white
Confirmacion 'quality' encontrada en ambos datasets
```

```
============================================================
Red wine - shape: (1599, 12)
Variables / columnas:
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
   'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density
   ', 'pH', 'sulphates', 'alcohol', 'quality']

Primeras 5 filas:
    fixed acidity   volatile acidity    citric acid residual sugar
0             7.4               0.70           0.00            1.9
1             7.8               0.88           0.00            2.6
2             7.8               0.76           0.04            2.3
3            11.2               0.28           0.56            1.9
4             7.4               0.70           0.00            1.9
    chlorides   free sulfur dioxide total sulfur dioxide    density
0       0.076                  11.0                 34.0     0.9978
1       0.098                  25.0                 67.0     0.9968
2       0.092                  15.0                 54.0     0.9970
3       0.075                  17.0                 60.0     0.9980
4       0.076                  11.0                 34.0     0.9978
     pH  sulphates   alcohol quality
0  3.51       0.56       9.4       5
1  3.20       0.68       9.8       5
2  3.26       0.65       9.8       5
3  3.16       0.58       9.8       6
4  3.51       0.56       9.4       5

Resumen numerico (describe):
                             count       mean        std      min
fixed acidity               1599.0   8.319637   1.741096  4.60000
volatile acidity            1599.0   0.527821   0.179060  0.12000
citric acid                 1599.0   0.270976   0.194801  0.00000
residual sugar              1599.0   2.538806   1.409928  0.90000
chlorides                   1599.0   0.087467   0.047065  0.01200
free sulfur dioxide         1599.0  15.874922  10.460157  1.00000
total sulfur dioxide        1599.0  46.467792  32.895324  6.00000
density                     1599.0   0.996747   0.001887  0.99007
pH                          1599.0   3.311113   0.154386  2.74000
sulphates                   1599.0   0.658149   0.169507  0.33000
alcohol                     1599.0  10.422983   1.065668  8.40000
quality                     1599.0   5.636023   0.807569  3.00000
                               25%       50%        75%       max
fixed acidity               7.1000   7.90000   9.200000  15.90000
volatile acidity            0.3900   0.52000   0.640000   1.58000
citrid acid                 0.0900   0.26000   0.420000   1.00000
residual sugar              1.9000   2.20000   2.600000  15.50000
```

```
chlorides                0.0700  0.07900  0.090000   0.61100
free sulfur dioxide      7.0000 14.00000 21.000000  72.00000
total sulfur dioxide    22.0000 38.00000 62.000000 289.00000
density                  0.9956  0.99675  0.997835   1.00369
pH                       3.2100  3.31000  3.400000   4.01000
sulphates                0.5500  0.62000  0.730000   2.00000
alcohol                  9.5000 10.20000 11.100000  14.90000
quality                  5.0000  6.00000  6.000000   8.00000

Distribucion de 'quality' (conteos):
quality
3     10
4     53
5    681
6    638
7    199
8     18
============================================================

============================================================
White wine - shape: (4898, 12)
Variables / columnas:
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
   'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density
   ', 'pH', 'sulphates', 'alcohol', 'quality']

Primeras 5 filas:
    fixed acidity   volatile acidity    citric acid residual sugar
0             7.0               0.27           0.36           20.7
1             6.3               0.30           0.34            1.6
2             8.1               0.28           0.40            6.9
3             7.2               0.23           0.32            8.5
4             7.2               0.23           0.32            8.5
    chlorides   free sulfur dioxide total sulfur dioxide    density
0       0.045                  45.0                170.0     1.0010
1       0.049                  14.0                132.0     0.9940
2       0.050                  30.0                 97.0     0.9951
3       0.058                  47.0                186.0     0.9956
4       0.058                  47.0                186.0     0.9956
     pH  sulphates   alcohol quality
0  3.00       0.45       8.8       6
1  3.30       0.49       9.5       6
2  3.26       0.44      10.1       6
3  3.19       0.40       9.9       6
4  3.19       0.40       9.9       6

```

```
Resumen numerico (describe):
                              count        mean        std       min
fixed acidity                4898.0    6.854788   0.843868   3.80000
volatile acidity             4898.0    0.278241   0.100795   0.08000
citric acid                  4898.0    0.334192   0.121020   0.00000
residual sugar               4898.0    6.391415   5.072058   0.60000
chlorides                    4898.0    0.045772   0.021848   0.00900
free sulfur dioxide          4898.0   35.308085  17.007137   2.00000
total sulfur dioxide         4898.0  138.360657  42.498065   9.00000
density                      4898.0    0.994027   0.002991   0.98711
pH                           4898.0    3.188267   0.151001   2.72000
sulphates                    4898.0    0.489847   0.114126   0.22000
alcohol                      4898.0   10.514267   1.230621   8.00000
quality                      4898.0    5.877909   0.885639   3.00000
                                25%       50%        75%        max
fixed acidity              6.300000   6.80000     7.3000   14.20000
volatile acidity           0.210000   0.26000     0.3200    1.10000
citrid acid                0.270000   0.32000     0.3900    1.66000
residual sugar             1.700000   5.20000     9.9000   65.80000
chlorides                  0.036000   0.04300     0.0500    0.34600
free sulfur dioxide       23.000000  34.00000    46.0000  289.00000
total sulfur dioxide     108.000000 134.00000   167.0000  440.00000
density                    0.991723   0.99374     0.9961    1.03898
pH                         3.090000   3.18000     3.2800    3.82000
sulphates                  0.410000   0.47000     0.5500    1.08000
alcohol                    9.500000  10.40000    11.4000   14.20000
quality                    5.0000    6.00000   6.000000    9.00000

Distribucion de 'quality' (conteos):
quality
3      20
4     163
5    1457
6    2198
7     880
8     175
9       5
============================================================
```

```
# ==========================================
# Red Bayesiana Gaussiana (Linear Gaussian BN)
# Codigo completo unificado
# ==========================================

!pip install pgmpy==1.0.0 --quiet

import numpy as np
```

```
import pandas as pd
import networkx as nx
import pickle
from typing import Optional
from pgmpy.estimators import HillClimbSearch
from pgmpy.models import LinearGaussianBayesianNetwork
from pgmpy.factors.continuous import LinearGaussianCPD
from sklearn.linear_model import LinearRegression

# --------------------------------
# Funcion: aprender estructura LGBN
# --------------------------------
def learn_lgbn_structure(data: pd.DataFrame,
                         target: str = 'quality',
                         score: str = 'bic',
                         max_parents: Optional[int] = None,
                         tabu_length: int = 100,
                         max_iter: int = 10000,
                         random_state: Optional[int] = None) ->
                            LinearGaussianBayesianNetwork:
    """
    Aprende la estructura (DAG) de una Linear Gaussian Bayesian Network
        desde `data`.
    Corrige nombres de columnas para evitar problemas con patsy/
       statsmodels.
    Devuelve un LinearGaussianBayesianNetwork con CPDs estimadas por
       regresion OLS.
    """
    assert isinstance(data, pd.DataFrame)

    safe_columns = {col: col.replace(" ", "_") for col in data.columns}
    data = data.rename(columns=safe_columns)
    target = safe_columns.get(target, target)

    variables = list(data.columns)
    score = score.lower()
    if score not in ('bic', 'aic'):
        raise ValueError("score debe ser 'bic' o 'aic'")
    scoring_method_name = 'bic-g' if score == 'bic' else 'aic-g'

    hc = HillClimbSearch(data)
    est_kwargs = {}
    if max_parents is not None:
        est_kwargs['max_indegree'] = int(max_parents)
    est_kwargs['tabu_length'] = int(tabu_length)
    est_kwargs['max_iter'] = int(max_iter)
```

```
51
52      best_model = hc.estimate(scoring_method=scoring_method_name, **
            est_kwargs)
53      lgbn = LinearGaussianBayesianNetwork(best_model.edges())
54
55      cpds = []
56      for node in lgbn.nodes():
57          parents = list(lgbn.predecessors(node))
58          y = data[node].values.reshape(-1, 1)
59          if len(parents) == 0:
60              mu = float(np.mean(y))
61              sigma = float(np.std(y, ddof=1))
62              beta = np.array([mu])
63              cpds.append(LinearGaussianCPD(variable=node, beta=beta, std
                    =sigma, evidence=[]))
64          else:
65              X = data[parents].values
66              reg = LinearRegression(fit_intercept=True)
67              reg.fit(X, y.ravel())
68              beta = reg.coef_.ravel()
69              intercept = float(reg.intercept_)
70              preds = reg.predict(X).reshape(-1,1)
71              residuals = y - preds
72              sigma = float(np.std(residuals, ddof=1))
73              beta_with_intercept = np.concatenate(([intercept], beta))
74              cpds.append(LinearGaussianCPD(variable=node, beta=
                    beta_with_intercept, std=sigma, evidence=parents))
75
76      lgbn.add_cpds(*cpds)
77
78      lgbn._learned_from_data = True
79      lgbn._training_columns = variables
80      lgbn._target = target
81      return lgbn
82
83  # --------------------------------
84  # Funcion: imprimir CPDs (texto)
85  # --------------------------------
86  def print_lgbn_cpds(lgbn: LinearGaussianBayesianNetwork):
87      """
88      Imprime las CPDs lineales gaussianas en formato legible.
89      """
90      print("\n=== CPDs lineales gaussianas ===")
91      cpds = getattr(lgbn, 'cpds', None) or lgbn.get_cpds()
92      for cpd in cpds:
93          var = cpd.variable
```

```
94          evidence = list(cpd.evidence) if cpd.evidence is not None else
                []
95          beta = np.asarray(cpd.beta) if cpd.beta is not None else np.
                array([])
96          std = float(cpd.std)
97          if len(evidence) == 0:
98              print(f"{var} ~ N({beta[0]:.4f}, {std:.4f})  (no parents)")
99          else:
100             intercept = float(beta[0])
101             coefs = beta[1:]
102             terms = " + ".join([f"{coefs[i]:.4f}*{evidence[i]}" for i
                    in range(len(evidence))])
103             print(f"{var} | {', '.join(evidence)}  =>  N({intercept:.4f
                    } + {terms}, {std:.4f})")
104
105 # -------------------------------
106 # Nueva funcion: mostrar tabla Nodo-Padres
107 # -------------------------------
108 def show_structure_table(lgbn: LinearGaussianBayesianNetwork) -> pd.
        DataFrame:
109     """
110     Devuelve una tabla con dos columnas: Nodo y Padres
111     """
112     rows = []
113     for node in lgbn.nodes():
114         parents = list(lgbn.predecessors(node))
115         parent_str = ", ".join(parents) if parents else "(ninguno)"
116         rows.append({"Nodo": node, "Padres": parent_str})
117     return pd.DataFrame(rows)
118
119 # -------------------------------
120 # Funcion: guardar red entrenada
121 # -------------------------------
122 def save_lgbn(lgbn: LinearGaussianBayesianNetwork, filename: str):
123     with open(filename, 'wb') as f:
124         pickle.dump(lgbn, f)
125     print(f"Modelo guardado en {filename}")
126
127 # -------------------------------
128 # Generador de datos sinteticos
129 # -------------------------------
130 def generate_synthetic_lgbn(n: int = 12,
131                             D: int = 3,
132                             N: int = 1000,
133                             target_idx: int = 0,
134                             max_parents: int = 3,
```

```
                                random_state: Optional[int] = None):
    """
    Genera un dataset sintetico y una LGBN sencilla.
    """
    rng = np.random.RandomState(random_state)
    names = [f"X{i}" for i in range(n)]
    edges = []
    for i in range(n):
        for j in range(i+1, n):
            if rng.rand() < 0.12:
                cur_parents = sum(1 for (u,v) in edges if v == names[j
                    ])
                if cur_parents < max_parents:
                    edges.append((names[i], names[j]))

    lgbn = LinearGaussianBayesianNetwork(edges)
    cpds = []
    for node in lgbn.nodes():
        parents = list(lgbn.predecessors(node))
        if len(parents) == 0:
            mu = rng.normal(0,1)
            sigma = float(rng.uniform(0.5, 2.0))
            cpds.append(LinearGaussianCPD(variable=node, beta=np.array
                ([mu]), std=sigma, evidence=[]))
        else:
            intercept = float(rng.normal(0,1))
            coefs = rng.normal(0,1,size=len(parents))
            sigma = float(rng.uniform(0.5, 2.0))
            beta = np.concatenate(([intercept], coefs))
            cpds.append(LinearGaussianCPD(variable=node, beta=beta, std
                =sigma, evidence=parents))
    lgbn.add_cpds(*cpds)

    def sample_once():
        sample = {}
        topo = list(nx.topological_sort(nx.DiGraph(lgbn.edges())))
        for node in topo:
            cpd = next(c for c in lgbn.cpds if c.variable == node)
            parents = list(cpd.evidence) if cpd.evidence is not None
                else []
            beta = np.asarray(cpd.beta)
            if len(parents) == 0:
                mean = float(beta[0])
            else:
                intercept = float(beta[0])
                coefs = beta[1:]
```

```
                parent_vals = np.array([sample[p] for p in parents])
                mean = intercept + float(np.dot(coefs, parent_vals))
            val = rng.normal(loc=mean, scale=float(cpd.std))
            sample[node] = float(val)
        return sample

    rows = [sample_once() for _ in range(N)]
    df = pd.DataFrame(rows, columns=names)
    return df, lgbn, names[target_idx]
```

```
# -------------------------------
# Ejemplo de uso
# -------------------------------

# 1) Aprender red para red_data (cargado en otra celda)
lgbn_red = learn_lgbn_structure(red_data, target='quality', score='bic'
    , max_parents=3, tabu_length=50, max_iter=100)
print("Estructura aprendida (aristas):", list(lgbn_red.edges()))
print_lgbn_cpds(lgbn_red)
print(show_structure_table(lgbn_red))
save_lgbn(lgbn_red, 'lgbn_red.pkl')

# 2) Generar datos sinteticos
data_syn, lgbn_syn, target_syn = generate_synthetic_lgbn(n=12, N=5000,
    target_idx=0,max_parents=3, random_state=123 34 )
print("Datos sinteticos shape:", data_syn.shape)
print("Target:", target_syn)
print("Estructura:", list(lgbn_syn.edges()))
print_lgbn_cpds(lgbn_syn)
print(show_structure_table(lgbn_syn))
```

```
Estructura aprendida (aristas): [('fixed_acidity', 'citric_acid'), ('
    fixed_acidity', 'density'), ('fixed_acidity', 'volatile_acidity'),
    ('fixed_acidity', 'residual_sugar'), ('citric_acid', '
    volatile_acidity'), ('citric_acid', 'alcohol'), ('volatile_acidity',
     'quality'), ('volatile_acidity', 'sulphates'), ('residual_sugar', '
    density'), ('quality', 'alcohol'), ('quality', 'sulphates'), ('
    alcohol', 'density'), ('alcohol', 'residual_sugar'), ('chlorides', '
    sulphates'), ('chlorides', 'pH'), ('chlorides', 'volatile_acidity'),
     ('chlorides', 'citric_acid'), ('chlorides', 'fixed_acidity'), ('
    chlorides', 'quality'), ('pH', 'fixed_acidity'), ('pH', 'alcohol'),
    ('pH', 'total_sulfur_dioxide'), ('free_sulfur_dioxide', '
    total_sulfur_dioxide'), ('free_sulfur_dioxide', 'pH'), ('
    total_sulfur_dioxide', 'fixed_acidity'), ('total_sulfur_dioxide', '
    residual_sugar'), ('total_sulfur_dioxide', 'quality'), ('
    total_sulfur_dioxide', 'citric_acid')]

```

```
=== CPDs lineales gaussianas ===
fixed_acidity | chlorides, total_sulfur_dioxide, pH
=>  N(35.7642 + -3.2880*chlorides + -0.0083*total_sulfur_dioxide + -8.0854*pH, 1.2323)
citric_acid | fixed_acidity, chlorides, total_sulfur_dioxide
=>  N(-0.4316 + 0.0751*fixed_acidity + 0.5628*chlorides + 0.0006*total_sulfur_dioxide, 0.1402)
density | fixed_acidity, residual_sugar, alcohol
=>  N(0.9990 + 0.0007*fixed_acidity + 0.0004*residual_sugar + -0.0008*alcohol, 0.0010)
volatile_acidity | fixed_acidity, citric_acid, chlorides
=>  N(0.4568 + 0.0231*fixed_acidity + -0.6824*citric_acid + 0.7288*chlorides, 0.1428)
residual_sugar | fixed_acidity, total_sulfur_dioxide, alcohol
=>  N(-0.3270 + 0.1200*fixed_acidity + 0.0103*total_sulfur_dioxide + 0.1332*alcohol, 1.3596)
quality | volatile_acidity, chlorides, total_sulfur_dioxide
=>  N(6.8453 + -1.6817*volatile_acidity + -1.6961*chlorides + -0.0037*total_sulfur_dioxide, 0.7284)
sulphates | volatile_acidity, chlorides, quality
=>  N(0.3577 + -0.1869*volatile_acidity + 1.4863*chlorides + 0.0477*quality, 0.1456)
alcohol | citric_acid, pH, quality
=>  N(-0.8532 + 1.0301*citric_acid + 2.3042*pH + 0.5975*quality, 0.8884)
chlorides ~ N(0.0875, 0.0471)  (no parents)
pH | chlorides, free_sulfur_dioxide
=>  N(3.3704 + -0.8707*chlorides + 0.0011*free_sulfur_dioxide, 0.1485)
free_sulfur_dioxide ~ N(15.8749, 10.4602)  (no parents)
total_sulfur_dioxide | free_sulfur_dioxide, pH
=>  N(93.1960 + 2.1249*free_sulfur_dioxide + -24.3004*pH, 24.2016)
                Nodo                                                Padres
0      fixed_acidity                chlorides, total_sulfur_dioxide, pH
1        citric_acid    fixed_acidity, chlorides, total_sulfur_dioxide
2            density            fixed_acidity, residual_sugar, alcohol
3   volatile_acidity             fixed_acidity, citric_acid, chlorides
4     residual_sugar      fixed_acidity, total_sulfur_dioxide, alcohol
5            quality volatile_acidity, chlorides, total_sulfur_dioxide
6          sulphates              volatile_acidity, chlorides, quality
7            alcohol                          citric_acid, pH, quality
8          chlorides                                           (ninguno)
9                 pH                   chlorides, free_sulfur_dioxide
10  free_sulfur_dioxide                                           (ninguno)
11 total_sulfur_dioxide                        free_sulfur_dioxide, pH

Modelo guardado en lgbn_red.pkl
```

```
Datos sinteticos shape: (5000, 12)
Target: X0
Estructura: [('X0', 'X1'), ('X0', 'X3'), ('X0', 'X7'), ('X1', 'X5'), ('
    X5', 'X6'), ('X5', 'X8'), ('X2', 'X6')]

=== CPDs lineales gaussianas ===
X0 ~ N(-0.3620, 1.1112)  (no parents)
X1 | X0  =>  N(1.6483 + -0.3755*X0, 1.3723)
X3 | X0  =>  N(0.8844 + -0.5342*X0, 1.6857)
X7 | X0  =>  N(1.4953 + 2.2216*X0, 1.3288)
X5 | X1  =>  N(1.6985 + 1.2122*X1, 0.6888)
X2 ~ N(-0.7551, 0.6408)  (no parents)
X6 | X2, X5  =>  N(1.8151 + 1.6593*X2 + 0.0882*X5, 1.0146)
X8 | X5  =>  N(-0.1994 + -0.5881*X5, 1.0937)
  Nodo     Padres
0   X0  (ninguno)
1   X1         X0
2   X3         X0
3   X7         X0
4   X5         X1
5   X2  (ninguno)
6   X6     X2, X5
7   X8         X5
```

```python
import numpy as np
import pandas as pd
import time
import networkx as nx

# ----------------------------
# Markov Blanket manual
# ----------------------------
def markov_blanket_manual(lgbn, target):
    """
    Calcula el entorno de Markov de un nodo en una LGBN.
    MB = padres U hijos U co-padres (otros padres de los hijos).
    """
    G = nx.DiGraph(lgbn.edges())
    parents = set(G.predecessors(target))
    children = set(G.successors(target))
    co_parents = set()
    for c in children:
        co_parents |= set(G.predecessors(c))
    return parents | children | co_parents

# ----------------------------
# Auxiliar: obtener CPD por nombre de variable
```

```
# ----------------------------
def _get_cpd_for_var(lgbn, var):
    cpds = getattr(lgbn, 'cpds', None) or lgbn.get_cpds()
    for c in cpds:
        if c.variable == var:
            return c
    raise KeyError(f"CPD para variable '{var}' no encontrada en la red.")

# ----------------------------
# Nuevo calculo SHAP segun algoritmo propuesto
# ----------------------------
def compute_shap_values(lgbn, data, target, instance: pd.Series):
    """
    Implementa el algoritmo SHAP propuesto (version directa que trabaja sobre MB(Y)):
      - Extrae padres/hijos y betas/std desde las CPDs lineales-gaussianas.
      - Calcula los phi segun el algoritmo y devuelve w* (x - mu) por variable en MB.
    Retorna: list[(var, shap_value), ...], stats dict {'time_sec', 'n_vars'}.
    """
    start = time.time()

    # 1) Calcular MB y lista de variables X (orden determinista)
    MB = sorted(list(markov_blanket_manual(lgbn, target)))
    S = [target] + MB  # orden: target primero
    X_vars = MB[:]      # variables explicativas para salida

    # 2) Inicializar phi para variables del MB (a 0)
    phi = {v: 0.0 for v in MB}

    # 3) Obtener CPD de Y y su varianza (sigma^2)
    cpd_Y = _get_cpd_for_var(lgbn, target)
    sigma_Y = float(cpd_Y.std)
    if sigma_Y == 0:
        raise ValueError(f"Desconocida/0 desviacion estandar para la variable objetivo {target}")
    inv_sigma2_Y = 1.0 / (sigma_Y ** 2)

    # phi_Y seun algoritmo (escala)
    phi_Y = inv_sigma2_Y

    # 4) Iterar hijos de Y
    G = nx.DiGraph(lgbn.edges())
```

```
    children_of_Y = list(G.successors(target))
    cpds = getattr(lgbn, 'cpds', None) or lgbn.get_cpds()

    for child in children_of_Y:
        # saltar hijos que no estan en la red de CPDs por seguridad
        try:
            cpd_child = _get_cpd_for_var(lgbn, child)
        except KeyError:
            continue

        sigma_child = float(cpd_child.std)
        if sigma_child == 0:
            continue
        inv_sigma2_child = 1.0 / (sigma_child ** 2)

        # extraer beta_{child, parent} (cpd.beta tiene intercept en pos 0)
        evidence = list(cpd_child.evidence) if cpd_child.evidence is not None else []
        beta = np.asarray(cpd_child.beta) if cpd_child.beta is not None else np.array([0.0])
        # si no aparece target en los padres, la contribucion directa es nula
        if target in evidence:
            idx_y = evidence.index(target)
            beta_child_y = float(beta[1 + idx_y])  # coeficiente correspondiente a Y
        else:
            beta_child_y = 0.0

        # actualizar phi del hijo (si pertenece a MB)
        if child in phi:
            phi[child] -= inv_sigma2_child * beta_child_y

        # propagar a los padres de child
        for j, parent in enumerate(evidence):
            beta_child_parent = float(beta[1 + j])
            # solo nos interesa actualizar padres que esten en MB
            # nota: si parent == target, entonces sera tratado en la formula anterior/propagacion
            if parent == target:
                # si parent==Y ya consideramos su efecto via phi_Y
                phi_Y += inv_sigma2_child * beta_child_y * beta_child_parent
            else:
                if parent in phi:
```

```
                    phi[parent] += inv_sigma2_child * beta_child_y *
                        beta_child_parent

    # 5) Ajustar por padres de Y (segundo bucle)
    parents_of_Y = list(G.predecessors(target))
    # extraer betas de Y (si tiene padres)
    evidence_Y = list(cpd_Y.evidence) if cpd_Y.evidence is not None
        else []
    beta_Y = np.asarray(cpd_Y.beta) if cpd_Y.beta is not None else np.
        array([0.0])
    for j, parent in enumerate(evidence_Y):
        beta_Y_parent = float(beta_Y[1 + j])
        if parent in phi:
            phi[parent] -= inv_sigma2_Y * beta_Y_parent

    # 6) Calcular w = -phi / phi_Y para X in MB (phi_Y no debe ser 0)
    if phi_Y == 0:
        raise ValueError("phi_Y es cero, no es posible dividir para
            obtener w.")
    w_dict = {v: - (phi[v] / phi_Y) for v in MB}

    # 7) Calcular SHAP vals = w_i * (x_i - mu_i)
    mu = data[X_vars].mean().values
    # alinear orden X_vars
    x_inst = instance[X_vars].values
    shap_vals = []
    for i, var in enumerate(X_vars):
        w_i = w_dict.get(var, 0.0)
        val = np.float64(w_i * (x_inst[i] - mu[i]))
        shap_vals.append((var, val))

    elapsed = time.time() - start
    stats = {"time_sec": elapsed, "n_vars": len(X_vars)}
    return shap_vals, stats

# -------------------------------
# Funcion para alinear nombres de columnas
# -------------------------------
def align_names(lgbn, df: pd.DataFrame) -> pd.DataFrame:
    """
    Asegura que los nombres de las columnas del DataFrame coincidan con
        los nodos de la red.
    """
    mapping = {}
    for col in df.columns:
        safe_col = col.replace(" ", "_")
```

```
        if safe_col in lgbn.nodes():
            mapping[col] = safe_col
    if mapping:
        df = df.rename(columns=mapping)
        print("Se han renombrado columnas para coincidir con los nodos
            del modelo:", mapping)
    return df


# ----------------------------
# Ejemplo de uso
# ----------------------------
# Para red real (vino tinto, variable objetivo "quality"):

red_aligned = align_names(lgbn_red, red_data)
row_red = red_aligned.iloc[0]
shap_real, stats_real = compute_shap_values(lgbn_red, red_aligned,
    target="quality", instance=row_example)
print("SHAP valores (vino tinto):", shap_real)
print("Stats:", stats_real)

# Para red sintetica (target = target_syn, ej: "X0"):
row_syn = data_syn.iloc[0]
shap_syn, stats_syn = compute_shap_values(lgbn_syn, data_syn, target=
    target_syn, instance=row_syn)
print("SHAP valores (sintetico):", shap_syn)
print("Stats:", stats_syn)
```

```
Se han renombrado columnas para coincidir con los nodos del modelo: {'
    fixed acidity': 'fixed_acidity', 'volatile acidity': '
    volatile_acidity', 'citric acid': 'citric_acid', 'residual sugar': '
    residual_sugar', 'chlorides': 'chlorides', 'free sulfur dioxide': '
    free_sulfur_dioxide', 'total sulfur dioxide': 'total_sulfur_dioxide
    ', 'density': 'density', 'pH': 'pH', 'sulphates': 'sulphates', '
    alcohol': 'alcohol', 'quality': 'quality'}
SHAP valores (vino tinto): [('alcohol', np.float64(-0.3082181649643309)
    ), ('chlorides', np.float64(0.02147432083720618)), ('citric_acid',
    np.float64(0.09152126398925092)), ('density', np.float64
    (-0.011367331767531852)), ('free_sulfur_dioxide', np.float64
    (-0.025149875709804002)), ('pH', np.float64(-0.12751933050387337)),
    ('quality', np.float64(0.0)), ('sulphates', np.float64
    (-0.100319552823781)), ('total_sulfur_dioxide', np.float64
    (0.049819227858875395)), ('volatile_acidity', np.float64
    (-0.18957205341206065))]
Stats: {'time_sec': 0.003373384475708008, 'n_vars': 10}

SHAP valores (sintetico): [('X0', np.float64(-0.0)), ('X1', np.float64
```

```
    (0.02148001295867983)), ('X3', np.float64(0.12743131618546408)), ('
    X7', np.float64(0.8807951667695173))]
Stats: {'time_sec': 0.002032756805419922, 'n_vars': 4}
```

```
import matplotlib.pyplot as plt
import numpy as np

# --------------------------------
# Comparacion red vs white con SHAP (todas las variables)
# --------------------------------
def compare_red_white_shap_allvars(red_data, white_data, target="
    quality",
                                   max_parents_list=[2,3,4], score="bic
                                       ",
                                   tabu_length=50, max_iter=100):
    """
    Entrena redes con distintos max_parents en red y white wine,
    calcula SHAP y genera graficas comparativas de barras mostrando
        todas las variables.
    Si una variable no tiene valor SHAP -> se marca con una cruz en
        lugar de barra.
    """
    results = []

    for mp in max_parents_list:
        # 1) Aprender red para red wine
        lgbn_red = learn_lgbn_structure(red_data, target=target,
                                        score=score, max_parents=mp,
                                        tabu_length=tabu_length,
                                           max_iter=max_iter)
        red_aligned = align_names(lgbn_red, red_data)
        row_red = red_aligned.iloc[0]
        shap_red, stats_red = compute_shap_values(lgbn_red, red_aligned
            , target, row_red)

        # 2) Aprender red para white wine
        lgbn_white = learn_lgbn_structure(white_data, target=target,
                                          score=score, max_parents=mp,
                                          tabu_length=tabu_length,
                                             max_iter=max_iter)
        white_aligned = align_names(lgbn_white, white_data)
        row_white = white_aligned.iloc[0]
        shap_white, stats_white = compute_shap_values(lgbn_white,
            white_aligned, target, row_white)

        # 3) Incluir todas las variables posibles (de ambos datasets)
        vars_all = sorted(set(red_aligned.columns) | set(white_aligned.
```

```
            columns))
        shap_red_dict = dict(shap_red)
        shap_white_dict = dict(shap_white)

        shap_red_vals = [shap_red_dict.get(v, None) for v in vars_all]
        shap_white_vals = [shap_white_dict.get(v, None) for v in
            vars_all]

        results.append({
            "max_parents": mp,
            "vars": vars_all,
            "shap_red": shap_red_vals,
            "shap_white": shap_white_vals,
            "stats_red": stats_red,
            "stats_white": stats_white
        })

        # 4) Plot grafico
        x = np.arange(len(vars_all))
        width = 0.35

        fig, ax = plt.subplots(figsize=(12,6))

        # Barras y cruces red wine
        for i, val in enumerate(shap_red_vals):
            if val is None:
                ax.scatter(x[i] - width/2, 0, marker="x", color="red",
                    s=60, label="_nolegend_")
            else:
                ax.bar(x[i] - width/2, val, width, color="red", alpha
                    =0.7, label="_nolegend_")

        # Barras y cruces white wine
        for i, val in enumerate(shap_white_vals):
            if val is None:
                ax.scatter(x[i] + width/2, 0, marker="x", color="gray",
                     s=60, label="_nolegend_")
            else:
                ax.bar(x[i] + width/2, val, width, color="gray", alpha
                    =0.7, label="_nolegend_")

        # Labels y titulo
        ax.set_ylabel("SHAP value")
        ax.set_title(f"Comparacion SHAP - max_parents={mp}")
        ax.set_xticks(x)
        ax.set_xticklabels(vars_all, rotation=45, ha="right")
```

```
        # Leyenda manual
        ax.bar(0, 0, color="red", alpha=0.7, label="Red Wine")
        ax.bar(0, 0, color="gray", alpha=0.7, label="White Wine")
        ax.scatter([], [], marker="x", color="red", label="Red Wine (
            sin valor)")
        ax.scatter([], [], marker="x", color="gray", label="White Wine
            (sin valor)")
        ax.legend()
        # Coste computacional debajo
        textstr = (f"Red Wine -> time: {stats_red['time_sec']:.4f}s,
            n_vars: {stats_red['n_vars']}\n"
                   f"White Wine -> time: {stats_white['time_sec']:.4f}s
                       , n_vars: {stats_white['n_vars']}")
        plt.figtext(0.5, -0.08, textstr, wrap=True, ha="center",
            fontsize=9)
        plt.tight_layout()
        plt.show()
    return results
# --------------------
# Ejemplo de uso
# --------------------
res = compare_red_white_shap_allvars(red_data, white_data,
                                     target="quality",
                                        max_parents_list=[3,4,5])
```

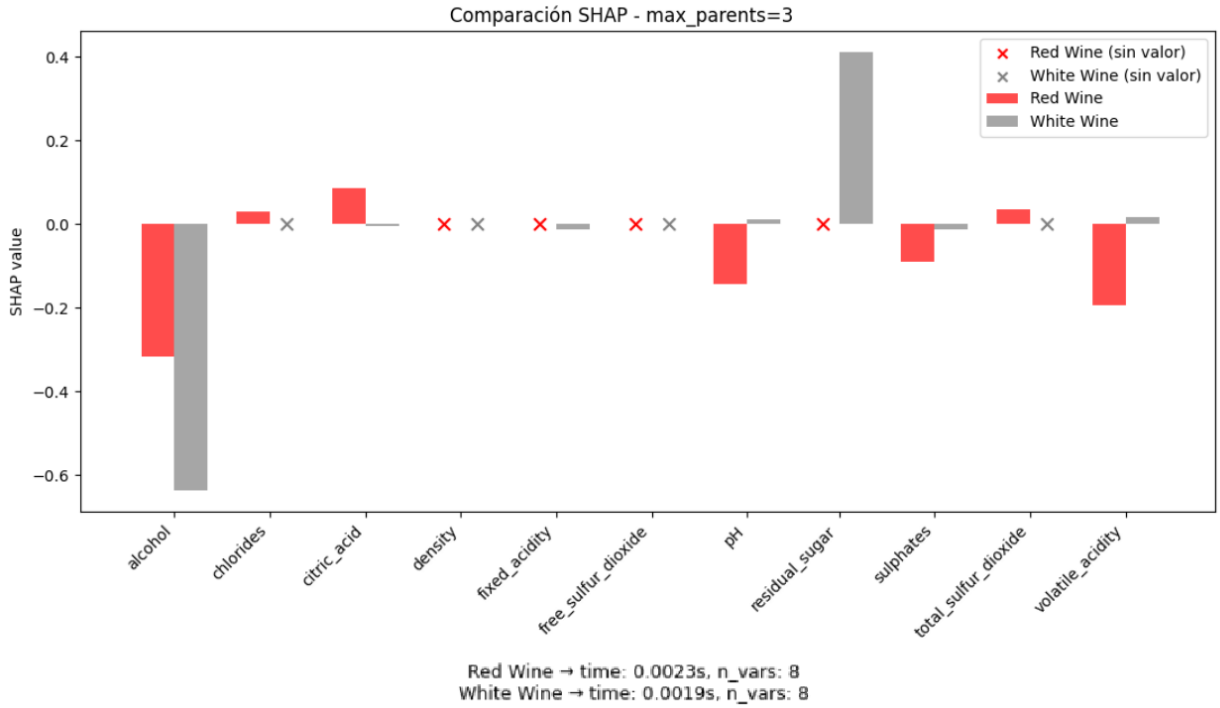


Figura II.1: Valores SHAP con RBG (max 3 padres) para vino tinto y blanco

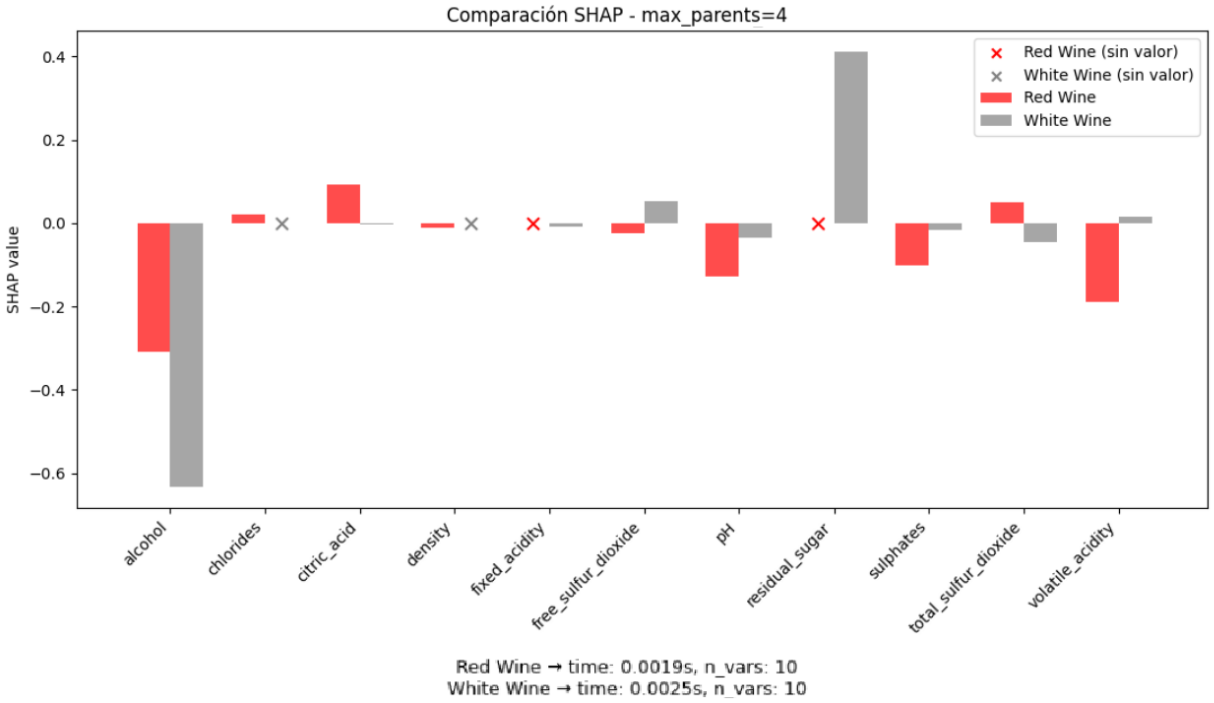


Figura II.2: Valores SHAP con RBG (max 4 padres) para vino tinto y blanco

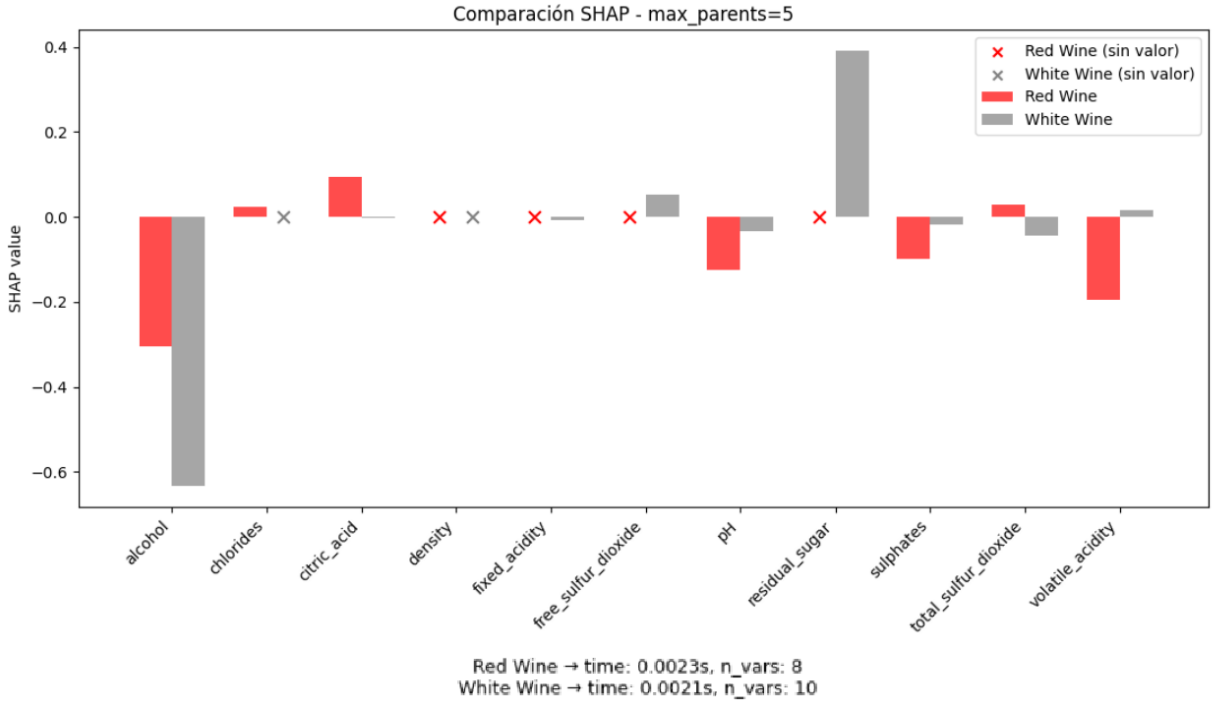


Figura II.3: Valores SHAP con RBG (max 5 padres) para vino tinto y blanco

```python
import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# ------------------------------------------------
# Auxiliar: asegurar que el target esta en la red
# ------------------------------------------------
def ensure_target_in_graph(lgbn, target_syn):
    """
    Asegura que el target este en los nodos de la red.
    Si no esta, intenta mapearlo al indice numerico de la variable.
    """
    nodes = list(lgbn.nodes())
    if target_syn in nodes:
        return target_syn
    # caso: target="X0" pero nodos son enteros o estan indexados
    if isinstance(target_syn, str) and target_syn.startswith("X"):
        idx = int(target_syn[1:])
        if idx < len(nodes):
            return nodes[idx]
    raise ValueError(f"Target {target_syn} no esta en los nodos de la red {nodes}")


# ------------------------------------------------
# Experimento de complejidad
# ------------------------------------------------
def complexity_experiment(n_list=[8,12,16,20], max_parents_list=[2,3,4,5],
                          N=5000, repeats=3, random_state=1234, return_df=True):
    """
    Realiza un experimento de coste computacional variando el numero de variables (n)
    y el parametro max_parents (que controla el tamano esperado del Markov Blanket).

    Devuelve una lista de resultados y opcionalmente un DataFrame.
    """
    results = []

    for n in n_list:
        for mp in max_parents_list:
            times = []
            mb_sizes = []
```

```
42
43              for r in range(repeats):
44                  # Generar datos y red sintetica
45                  data_syn, lgbn_syn, target_syn =
                        generate_synthetic_lgbn(
46                      n=n, N=N, target_idx=0, max_parents=mp,
47                      random_state=random_state + r + n*10 + mp*100
48                  )
49
50                  # Asegurar target valido
51                  target_syn = ensure_target_in_graph(lgbn_syn,
                        target_syn)
52
53                  # Tomamos una instancia
54                  row_syn = data_syn.iloc[0]
55
56                  # Medir tiempo
57                  start = time.time()
58                  shap_vals, stats = compute_shap_values(
59                      lgbn_syn, data_syn, target=target_syn, instance=
                            row_syn
60                  )
61                  end = time.time()
62
63                  times.append(end-start)
64                  mb_sizes.append(stats["n_vars"])
65
66              results.append({
67                  "n": n,
68                  "max_parents": mp,
69                  "time_mean": np.mean(times),
70                  "time_std": np.std(times),
71                  "mb_mean": np.mean(mb_sizes),
72                  "mb_std": np.std(mb_sizes),
73              })
74
75      if return_df:
76          return pd.DataFrame(results)
77      return results
78
79
80  # ------------------------------------------------
81  # Visualizacion de resultados
82  # ------------------------------------------------
83  def plot_complexity_results(df_results):
84      """
```

```
    Genera graficas de:
    - Tiempo medio de computo en funcion de n y max_parents
    - Tamano medio del MB en funcion de n y max_parents
    """
    fig, axes = plt.subplots(1, 2, figsize=(14,5))

    # --- Tiempo
    for mp in sorted(df_results["max_parents"].unique()):
        sub = df_results[df_results["max_parents"]==mp]
        axes[0].errorbar(sub["n"], sub["time_mean"], yerr=sub["time_std
            "],
                         marker="o", label=f"max_parents={mp}")
    axes[0].set_xlabel("Numero de variables (n)")
    axes[0].set_ylabel("Tiempo medio (s)")
    axes[0].set_title("Tiempo de computo vs n")
    axes[0].legend()

    # --- MB size
    for mp in sorted(df_results["max_parents"].unique()):
        sub = df_results[df_results["max_parents"]==mp]
        axes[1].errorbar(sub["n"], sub["mb_mean"], yerr=sub["mb_std"],
                         marker="s", label=f"max_parents={mp}")
    axes[1].set_xlabel("Numero de variables (n)")
    axes[1].set_ylabel("Tamano medio del MB")
    axes[1].set_title("Tamano del MB vs n")
    axes[1].legend()

    plt.tight_layout()
    plt.show()


# ------------------------------------------------
# Ejemplo de uso
# ------------------------------------------------
df_results = complexity_experiment(
    n_list=[15, 20, 25, 30, 35, 40],
    max_parents_list=[3, 5, 7, 9, 11],
    N=3000,
    repeats=5
)

# Mostrar tabla
print("=============== Resultados del experimento ===============")
print(df_results)

# Guardar tambien en una variable para usar despues
```

```
130 results_table = df_results.copy()
131
132 # Mostrar graficas
133 plot_complexity_results(df_results)
```

```
1  =============== Resultados del experimento ===============
2       n  max_parents  time_mean  time_std  mb_mean    mb_std
3  0   15            3   0.001620  0.000097      4.4  1.496663
4  1   15            5   0.001790  0.000291      4.6  2.576820
5  2   15            7   0.001723  0.000242      3.8  1.720465
6  3   15            9   0.001634  0.000115      4.2  1.166190
7  4   15           11   0.001645  0.000085      5.8  1.166190
8  5   20            3   0.001668  0.000054      5.8  2.561250
9  6   20            5   0.003853  0.004048      5.4  3.555278
10 7   20            7   0.001980  0.000364      6.6  1.854724
11 8   20            9   0.002318  0.000637      5.2  1.469694
12 9   20           11   0.001803  0.000288      4.4  1.743560
13 10  25            3   0.001790  0.000072      8.6  2.244994
14 11  25            5   0.002079  0.000355      7.6  3.006659
15 12  25            7   0.001810  0.000122      6.0  3.405877
16 13  25            9   0.001880  0.000192      5.6  2.416609
17 14  25           11   0.001919  0.000257      7.8  4.445222
18 15  30            3   0.001899  0.000238      7.2  1.833030
19 16  30            5   0.002164  0.000438      7.4  3.720215
20 17  30            7   0.002066  0.000222      7.6  2.870540
21 18  30            9   0.002852  0.001020      8.4  1.356466
22 19  30           11   0.002730  0.001624     11.0  4.774935
23 20  35            3   0.002014  0.000274      9.0  4.690416
24 21  35            5   0.002034  0.000066     15.4  2.870540
25 22  35            7   0.002537  0.000607     11.4  3.200000
26 23  35            9   0.002241  0.000432     12.6  4.498889
27 24  35           11   0.003686  0.003319     14.8  6.910861
28 25  40            3   0.002492  0.000466      9.0  3.033150
29 26  40            5   0.002276  0.000296     13.0  3.577709
30 27  40            7   0.002144  0.000063     13.8  3.544009
31 28  40            9   0.002152  0.000267      9.6  2.727636
32 29  40           11   0.002441  0.000409     13.4  4.409082
```