Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Explicabilidad basada en lógica para modelos simples de Inteligencia Artificial:

Sobre la complejidad y los algoritmos para el cálculo de *features* relevantes, necesarias y útiles en clasificadores booleanos

Tesis de Licenciatura en Ciencias de la Computación

Tomás Capdevielle

Director: Santiago Cifuentes
Buenos Aires, 2025

Explicabilidad basada en lógica para modelos simples de Inteligencia Artificial:

Sobre la complejidad y los algoritmos para el cálculo de *features* relevantes, necesarias y útiles en clasificadores booleanos

Resumen Esta tesis propone una exploración sobre el campo de la explicabilidad en modelos de inteligencia artificial, comúnmente conocido como XAI. En concreto, el enfoque de este trabajo está puesto en la explicabilidad basada en lógica; esto es, dado un modelo de clasificación y una entrada para el mismo, seleccionar aquellas *features* o atributos que cumplan con ciertas propiedades lógicas. Esto da lugar a las nociones de *features* relevantes y necesarias, en cuyo cálculo se profundiza. En este sentido, se aportan resultados formales que sustenten algoritmos existentes en la literatura, así como también se proponen algoritmos eficientes para detectar la necesidad de *features* en modelos tanto simples (e.g. árboles de decisión) como complejos (e.g. redes neuronales). A su vez, se exploran generalizaciones para la noción de relevancia, con el objetivo de capturar una visión más detallada del comportamiento de un modelo para una predicción dada. Por último, se introduce una nueva noción global de *utilidad*, que apunta a explicar si una *feature* es importante para el comportamiento de un modelo a nivel general, i.e. sin considerar una entrada en particular. De aquí surge también una propuesta para un sistema de *feature ranking* basado en utilidad, para cuyo cálculo se presentan algoritmos eficientes para ciertos modelos.

Palabras clave: explicabilidad, inteligencia artificial, features relevantes, features necesarias, lógica.

Logic-based explainability for simple Artificial Intelligence models:

On the complexity and algorithms for computing relevant, necessary and useful features in boolean classifiers

Abstract This thesis presents an exploration of the field of explainability in artificial intelligence models, commonly known as XAI. Specifically, we focuse on logic-based explainability; that is, given a classification model and an input, the task of identifying those features that satisfy certain logical properties. This leads to the notions of relevant and necessary features, whose computation is studied in depth in this work. Our goal is to provide formal results that support existing algorithms in the literature, as well as to propose efficient algorithms for detecting feature necessity in both simple models (such as decision trees) and complex ones (such as neural networks). Additionally, we explore two different generalizations for the notion of relevancy, aiming to capture a more detailed understanding of a model's behavior for a given prediction. Finally, we introduce a new global notion of utility, which seeks to determine whether a feature is important for the overall behavior of a model on a general scope, that is without considering any specific input. Based on this notion, we also present the foundations for a feature ranking system based on utility, for which efficient computation algorithms are also proposed for certain models.

Keywords: explainability, artificial intelligence, feature relevancy, feature necessity, logic.

AGRADECIMIENTOS

Quiero agradecer primero a mi papá Alejandro y a mi mamá Vilma, por brindarme apoyo, cariño y acompañamiento por muchos años, tanto durante el proceso de este trabajo en particular, como durante mi cursada universitaria (y mi vida) en general. Nada de esto habría sido ni remotamente posible sin mis dos pilares. También agradezco al resto de mi familia, especialmente mis abuelas Yoli y Carmen por todo su amor.

A Alejo, Facundo y Santiago, por acompañarme (al principio en el día a día, luego más esporádicamente o a la distancia) durante más de la mitad de mi vida.

A Clarisa, que me ayudó con su apoyo, su atención a mis problemas, locuras, y desesperaciones, y su invaluable compañía.

A Haydée Toronchik, por bancarme durante horas y horas de charlas durante tanto tiempo, y por animarme a enfrentar las dificultades y tomar decisiones difíciles.

A quien fue mi profesor de matemática, Mauro Nicodemo, que me dijo hace más de diez años que me veía haciendo esta carrera.

A mi director Santiago, por proponerme un tema de tesis muy interesante, aceptarme para trabajar con él, y guiarme a través de todo el arduo proceso. También a Juan Kamienkowski, por llevar a cabo el Taller de Tesis en 2024, que me ayudó a volver a juntar fuerzas para dar este último paso en la carrera, y a encontrar un tema para trabajar.

A Santiago Figueira y Sergio Abriola, por aceptar ser jurados de este trabajo. También agradezco a Pablo Barenbaum (y le sumo un pedido de disculpas por no haber podido continuar con el trabajo que empezamos juntos) y a Sergio Abriola –nuevamente– por haber dedicado parte de su tiempo a presentarme posibles temas de tesis hace ya dos años.

A mis escuelas primaria y secundaria: Aletheia y el ILSE, por darme un espacio en el que sentirme cómodo y estimulado a aprender y formar mi propio criterio.

Y por último, pero no menos importante, a la universidad pública, que me abrió sus puertas cuando yo estaba en un mal momento, intentando superar una *crisis académica*. Este agradecimiento se extiende también a la Facultad de Ciencias Exactas y Naturales, al Departamento de Computación, y especialmente a todos los docentes cuyas clases tuve la suerte de presenciar, así como también a mis compañeros de cursada y trabajos prácticos (principalmente a Maxi, los Lucases, Gabriel y también la gente de *La Plebe*, que ayudó a que esos años de *cursada pandémica* sean más llevaderos y menos surrealistas).

A todas estas personas les digo que estoy enormemente agradecido por, de una u otra forma, contribuir a que esta tesis haya sido empezada y, sobre todo, terminada.

A Vilma y Alejandro.

Índice general

1. INTRODUCCIÓN

Motivación La inteligencia artificial está cada vez más presente en nuestra vida cotidiana, y en particular el uso de modelos basados en aprendizaje automático es más común que nunca [15, 21, 55]. La capacidad de estos últimos está alcanzando niveles inéditos [16, 49, 58, 66] gracias al aprovechamiento de grandes repositorios públicos de datos [70], y de hardware especializado para la etapa de aprendizaje [56]. Esto permite que las aplicaciones posibles sean cada vez más diversas, hasta el punto de que hacer uso de modelos de inteligencia artificial se ha vuelto algo corriente en la vida de cualquier persona.

Sin embargo, este incremento en poder, utilidad y popularidad también ha sacado a relucir cuestionamientos y preocupaciones respecto del uso de la IA, debido tanto al sesgo involucrado en los modelos, como a ciertos eventos catastróficos causados por estos, o incluso a las cuestiones éticas asociadas a su uso [2, 36, 63, 71]. Este trabajo tratará acerca de uno de estos aspectos: el de la explicabilidad sobre su comportamiento; esto se refiere a la capacidad de evaluar y justificar las decisiones tomadas por un modelo, y es, de hecho, uno de los puntos débiles de los modelos de aprendizaje automático más poderosos y utilizados en la actualidad [57, 65]. La complejidad de estos modelos ha crecido tanto que se ha vuelto cada vez más difícil comprender e interpretar su funcionamiento, hasta el punto en que es frecuente utilizarlos como *cajas negras*.

La explicabilidad es especialmente importante en contextos de uso en los que es necesario poder proveer al usuario de algún tipo de justificación y/o interpretación del resultado (o predicción) generado por el modelo. Esto es de vital relevancia cuando estas predicciones tienen consecuencias serias sobre la vida de un ser humano, como en contextos de medicina, finanzas o derecho. Por ejemplo:

- Si una persona solicita un crédito bancario y este le es rechazado por decisión de un modelo de inteligencia artificial usado por la institución financiera, entonces ésta debería ofrecer una justificación apropiada, convincente y entendible al solicitante (lo cual le permitiría, en particular, modificar su situación para poder acceder al crédito más adelante).

- Si un paciente recibe un diagnóstico médico asistido por un sistema automático, dicho sistema debería poder aportar una explicación fundamentada y con un lenguaje adecuado para que el profesional de la salud que atiende al paciente pueda explicarle detalladamente cuál es su situación, y cuáles son las medidas a tomar al respecto. Esto es importante porque un error podría acarrear consecuencias graves tanto en el ámbito legal, como en la salud del paciente.

Se puede afirmar entonces que la posibilidad de explicar las decisiones tomadas por estos modelos es uno de los aspectos esenciales para poder avanzar hacia un concepto de inteligencia artificial que cumpla con ciertos estándares de ética, que se han ido desarrollando en los últimos años como una hoja de ruta para poder aplicar este tipo de herramientas en la vida de las personas de una manera *segura* y *justa* [35, 31, 51]. Además, la explicabilidad no serviría solo para mejorar la seguridad en las aplicaciones de la IA, sino que también puede aprovecharse para mejorar la calidad y capacidad de los modelos, dado que al poder explicar sus resultados, se posibilita una mayor comprensión de los mismos.

Contexto y antecedentes A pesar de que las preocupaciones éticas por la interpretabilidad de los modelos de inteligencia artificial son relativamente recientes, sus bases teóricas existen desde hace varias décadas. Una de las propuestas más difundidas para este propósito es conocida como *feature ranking*; y consiste en asignarle un cierto valor a cada *feature* de un modelo, de acuerdo a su importancia para una predicción dada.

Los fundamentos teóricos en los que se basa el método más popular de *feature ranking* corresponden a una noción de la teoría de juegos: el *Shapley value* [67]. Este es un valor que surge en el contexto de la teoría de juegos cooperativos, y busca asignar *justamente* (es decir, de manera proporcional a cada contribución individual) el valor total obtenido por la colaboración de un conjunto de jugadores en un juego de equipo. La analogía con la explicabilidad en inteligencia artificial proviene de identificar a las *features* del modelo como jugadores, y a la tarea de predecir o clasificar una cierta entrada, como el juego sobre el cual estos colaboran. Así, el *Shapley value* asigna un número (una puntuación, o *score*) a cada jugador (*feature*) de acuerdo a su importancia en el resultado del juego (*predicción* del modelo para una cierta entrada), lo cual permite ordenarlos según su peso en relación al juego (la tarea de *explicar* el resultado).

Por un lado, el *Shapley value* cuenta con diversas propiedades axiomáticas deseables a efectos de la tarea de explicabilidad (cuya mención queda por fuera del alcance de este trabajo), las cuales hereda del formalismo sobre el cual se apoya. Pero por otro lado, es considerablemente costoso de calcular en términos computacionales [54, 4, 72]; por lo que se suele recurrir a aproximaciones (como el muy difundido SHAP score [50]), que no son especialmente precisas [32], y que además no mantienen todas las propiedades axiomáticas de la versión exacta. Además de esto, recientemente los métodos basados en *Shapley values* han recibido diferentes críticas; por ejemplo:

- En [42] se demuestra que los *Shapley values* pueden asignar un puntaje no nulo a *features irrelevantes* desde un punto de vista de lógico. Similarmente, pueden asignar un puntaje nulo a *features relevantes*. Además, estos problemas no se limitan a las aproximaciones de los *Shapley values* exactos, sino a los métodos basados en *Shapley values* en general.

- En [20, 46, 68] se exponen problemas (y algunas mitigaciones) con el SHAP score en cuanto a su sensibilidad a la distribución de los datos de entrenamiento del modelo, así como también su susceptibilidad al ruido.

Alternativamente, existe otro enfoque para producir explicaciones en el contexto de los modelos de inteligencia artificial: éste se suele denominar *feature selection*, y consiste en elegir un subconjunto de las *features* de un modelo, según algún criterio que apunte a capturar aquellas *features* que mejor expliquen una predicción, o el comportamiento del modelo. Una posible "instanciación" de *feature selection* es la que se trata de seleccionar *features* que satisfagan ciertas propiedades lógicas; es por eso que en la literatura se suele conocer a este enfoque como *explicabilidad basada en lógica* (o *logic-based explainability*) [52]. La idea es hallar conjuntos (idealmente minimales) de *features* que *determinen* por sí solos el resultado de una predicción por parte de un modelo para una entrada específica.

En el campo de la explicabilidad basada en lógica, los conceptos en los que se fundan las explicaciones se relacionan con la noción de *prime implicant*, que corresponde a conjuntos de variables proposicionales que determinan el valor de verdad de una fórmula. En este contexto, se acuñó el término de *sufficient reasons* para hacer referencia a estos conjuntos

de *features* que, si se dejan fijos, no se altera la clasificación final provista por el modelo, sin importar cómo o cuánto se modifique el valor del resto de sus *features*. Estos conjuntos permiten clasificar a las *features* como relevantes, irrelevantes o necesarias. A través de estas caracterizaciones, es posible decidir qué aspectos de la entidad dada como entrada al modelo fueron los más y menos determinantes para el resultado final; y esto a su vez es útil para poder brindar información acerca del comportamiento del modelo bajo análisis.

Este acercamiento a la explicabilidad basada en lógica, que es el que ocupa el interés del presente trabajo, ha tomado mayor inercia en los últimos años [24, 38], y se cimienta en herramientas tradicionales de la lógica proposicional; de esta manera, su objetivo es emplear estos formalismos para poder brindar un marco teórico riguroso, y una base precisa y bien definida a la tarea de la explicabilidad, de manera tal que sea posible contar con ciertas garantías teóricas (las cuales son especialmente importantes en contextos sensibles como los mencionados previamente). Debido a lo reciente del desarrollo de estas técnicas, se está evaluando activamente su utilidad respecto de la eficacia que puedan tener, así como también su factibilidad en cuanto a complejidad computacional.

En cuanto a referencias y literatura acerca de la explicabilidad basada en lógica, en [52] se realiza una exposición del panorama actual y los avances obtenidos en los últimos años. Se ha estudiado en diversos trabajos (algunos de ellos son [6, 8, 18, 40, 27]) la complejidad de los problemas de encontrar y enumerar explicaciones (y *sufficient reasons*) para modelos basados en diversas estructuras de representación particulares, obteniendo en ocasiones resultados positivos (problemas tratables) y en otros casos negativos (problemas intratables).

Objetivos En este trabajo, se buscará estudiar los conceptos de *sufficient reason*, *feature relevante* y *feature necesaria* mencionados, para modelos simples, que pueden variar desde árboles de decisión binarios, hasta circuitos booleanos[1]. Para algunos de estos modelos, se conocen actualmente ciertos resultados de complejidad computacional, a los cuales se hará referencia más adelante. Si bien estos modelos no son los más potentes o utilizados en la actualidad (como lo son las redes neuronales, o los *transformers*), sí ofrecen un nivel de precisión aceptable y útil en diversos contextos. Además, resultados recientes en el área [25, 53, 28, 41, 10] indican que es posible *compilar* modelos complejos en otros más sencillos de manera *offline*, a efectos de las tareas de explicabilidad. Una vez realizada la compilación, sería posible utilizar algoritmos más eficientes para los modelos sencillos y trasladarlos a familias de *predictores* más sofisticados, realizando las consultas de explicabilidad de manera *online*.

Es así que se propone profundizar en el área de la explicabilidad basada en lógica, ahondando en los problemas computacionales asociados al cálculo de explicaciones basadas en las nociones de sufficient reason y abductive explanation definidas en [52]. Dentro de este campo, concretamente se centrará la atención en estudiar el problema de decidir qué *features* de un modelo dado son *relevantes* para una predicción sobre una entidad. Este problema se puede plantear de diferentes maneras, y ese será uno de los puntos de partida que se abordará en siguientes secciones. Como se verá luego, el problema de decidir si una *feature* es *relevante* se relaciona con el problema de hallar las *sufficient reasons* de un modelo para una entidad en particular.

Además, se estudiará también algunas variantes y generalizaciones de los problemas de

[1] Como se verá más adelante, algunos de los resultados presentados pueden ser extendidos a modelos más complejos.

relevancia y necesidad de *features*, las cuales surgieron durante el transcurso de esta investigación. Por último, se introduce una nueva noción de *utilidad* de *features*, proponiendo un enfoque global al comportamiento de un modelo, y no limitado a una entrada y salida particulares. Esta noción también dará lugar a un sistema de *feature ranking* basado en utilidad.

Contribuciones Como parte de la revisión bibliográfica acerca de los conceptos fundamentales de la explicabilidad basada en lógica, se estudiará en profundidad el problema de detectar *features* relevantes en árboles de decisión. En este sentido, esta tesis profundiza en nociones y resultados presentados por otros autores, proponiendo formalizaciones para sustentar los algoritmos desarrollados por ellos. También se definirán dos generalizaciones para el problema de detección de relevancia, mostrando que ambas resultan NP-COMPLETE para árboles booleanos de decisión.

Por otro lado, se establece una frontera en la tratabilidad del problema de relevancia de *features*, dada por la relación entre este problema y el clásico problema de satisfacibilidad de modelos (comúnmente conocido como SAT). Esto permite definir límites más precisos a la hora de analizar la complejidad computacional del problema de detección de relevancia.

En cuanto al estudio de las *features* necesarias, se propone una caracterización útil a efectos algorítmicos para las mismas, la cual a su vez permite diseñar algoritmos eficientes para detectar necesidad en modelos complejos, para los cuales no se conocía una cota de complejidad. Esto extiende la frontera de tratabilidad del problema, mostrando que la necesidad de una *feature* puede ser decidida eficientemente para cualquier clasificador *razonable*. Además, se proponen algoritmos con complejidad lineal para calcular la totalidad de las *features* necesarias en árboles de decisión y otros modelos basados en diagramas de decisión.

Adicionalmente, se introduce una nueva noción global de importancia de *features*, llamada *utilidad*: este concepto apunta a describir el comportamiento general de un modelo, independientemente de una entrada particular. Luego, se prueba que la utilidad está relacionada con la relevancia y con la necesidad, además de mostrar una relación (en términos de complejidad computacional) entre el problema de equivalencia de modelos y la detección de utilidad de *features*. A su vez, se propone un sistema de puntuación de *features* en base a su utilidad, que puede ser calculado en tiempo cuadrático para árboles de decisión.

Por último, de manera simultánea a esta tesis, se desarrolló un trabajo [17] (disponible actualmente como *pre-impresión*), que contiene generalizaciones sobre los resultados de relevancia y necesidad de *features* para árboles de decisión con *features* categóricas y numéricas. Además, en [17] se incluye experimentación para el sistema de *feature ranking* definido en esta tesis (para estos árboles de decisión generalizados), con una complejidad cuadrática respecto del tamaño de entrada.

Organización del trabajo Esta tesis está estructurada de la siguiente manera: en el Capítulo 2 se presentan las definiciones de todos los conceptos que aparecerán a lo largo del trabajo, incluyendo elementos del contexto que rodea a la investigación (como nociones de lógica proposicional y clasificadores), conceptos centrales a la tesis (explicaciones formales, *sufficient reasons*, etc.), así como también contenido auxiliar que no tiene relación directa con el tema de la tesis, pero sí es de utilidad para demostrar resultados acerca del mismo.

Luego, en el Capítulo 3 se realiza un análisis de la complejidad computacional de varios problemas relacionados con la explicabilidad basada en lógica para clasificadores booleanos.

Se explora tanto la relevancia como la necesidad de *features*, y adicionalmente se estudia el concepto de utilidad introducido en este mismo trabajo.

Por último, en el Capítulo 4 se mencionan algunas conclusiones y potenciales direcciones de trabajo a futuro.

2. DEFINICIONES Y RESULTADOS PRELIMINARES

Se comenzará definiendo algunas nociones básicas de lógica proposicional que serán de utilidad para enmarcar los conceptos centrales de esta tesis.

2.1. Lógica proposicional

Una fórmula proposicional φ sobre las variables proposicionales (o átomos) $x_1, \ldots, x_n$ es una expresión de la forma x_i, o bien una conjunción ($\wedge$), disyunción ($\vee$) o negación ($\neg$) de otras fórmulas proposicionales; es decir, se forman según la gramática[1]:

$$\varphi ::= x_i \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

A lo largo del trabajo, se utilizarán letras griegas minúsculas para referirse a fórmulas proposicionales (por ejemplo: φ, ψ, etc). En cuanto a las variables, se utilizará la letra x con subíndices, o bien letras latinas minúsculas (en general x, y, z). Además, se denota $Var(\varphi)$ al conjunto de las variables proposicionales que aparecen en la fórmula φ.

El elemento básico del que se componen las fórmulas es el literal. Un *literal* es una variable x_i (literal positivo) o su negación $\neg x_i$ (a veces notado $\overline{x_i}$, literal negativo).

Ejemplo 2.1.1. Dada una fórmula $(x_1 \wedge x_2) \vee \neg x_3$, sus literales son x_1, x_2 y $\overline{x_3}$.

Trazando un paralelismo con el universo del aprendizaje automático y los modelos de inteligencia artificial, se suele asociar un literal positivo con una *feature* de valor 1, y a un literal negativo con una *feature* de valor 0.

Al combinar literales usando los conectivos mencionados, se puede obtener un *término* (conjunción de literales) o una *cláusula* (disyunción de literales). El tamaño de un término (o cláusula) se define como la cantidad de literales que lo (la) componen. Un *subtérmino* (o *subcláusula*) de un término t (cláusula c) es un término (cláusula) que utiliza un subconjunto de las variables que aparecen en t (c).

Ejemplo 2.1.2. Son términos de tamaños 1, 3 y 2 respectivamente:

- x_1
- $x_1 \wedge x_2 \wedge x_3$
- $x_1 \wedge \neg x_2$

Son cláusulas de tamaños 1, 3 y 2 respectivamente:

- x_1
- $x_1 \vee x_2 \vee x_3$
- $x_1 \vee \neg x_2$

Dada una fórmula proposicional φ y una asignación (o valuación) $\mathbf{x} : Var(\varphi) \to \{0, 1\}$ para sus variables, se dice que $\mathbf{x}$ satisface φ cuando $\varphi(\mathbf{x}) = 1$, donde la evaluación de las fórmulas en las asignaciones tiene la interpretación habitual (hacer verdadera la expresión que resulta al reemplazar las variables por los valores indicados en la asignación).

Se dice que un término t *implica* a una fórmula φ (notado $t \models \varphi$) cuando cualquier asignación a las variables de t que satisfaga a t, también satisface a φ.

[1] Los paréntesis se omitirán por simplicidad cuando la interpretación de la fórmula sea clara sin ellos.

Asimismo, al combinar términos y cláusulas se pueden formar fórmulas con cierta forma especial: una fórmula está en *forma normal disyuntiva* (o DNF, por disjunctive normal form) cuando es una disyunción de términos; análogamente, una fórmula está en *forma normal conjuntiva* (o CNF, por conjunctive normal form) cuando es una conjunción de cláusulas. El tamaño de una fórmula en DNF (o CNF) se define como la sumatoria de los tamaños de cada uno de sus términos (cláusulas).

Ejemplo 2.1.3. Son fórmulas en DNF:

- x_1
- $x_1 \wedge x_2 \wedge x_3$
- $(x_1 \wedge \neg x_2) \vee x_3$
- $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_4)$

Son fórmulas en CNF:

- x_1
- $x_1 \vee x_2$
- $(x_1 \vee \neg x_2) \wedge x_3$
- $(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3)$

Además, se dice que una fórmula está en n-DNF (resp. n-CNF) si está en DNF (CNF) y cada uno de sus términos (cláusulas) tiene a lo sumo n literales. Por otro lado, cierto tipo de fórmulas serán útiles más adelante, y se trata de las fórmulas monótonas:

Definición 1 (Fórmula monótona)**.** Una fórmula φ se dice *monótona* si para cada variable, todos los literales donde aparece tienen la misma *polaridad* (son todos o positivos, o bien negativos).

Ejemplo 2.1.4. La fórmula $(x_1 \vee x_2 \vee \neg x_3) \wedge x_1 \wedge \neg x_3$ es monótona, pero la fórmula $(x_1 \vee x_2 \vee \neg x_3) \wedge \neg x_1 \wedge x_3$ no lo es, pues x_1 aparece con y sin negación, al igual que x_3.

Se dice que una fórmula φ *implica* una fórmula ψ (y se nota $\varphi \models \psi$) cuando todas las asignaciones a variables que satisfacen a φ también satisfacen a ψ. Además, dos fórmulas φ y ψ son *equivalentes* (notado $\varphi \equiv \psi$) si $\varphi \models \psi$ y $\psi \models \varphi$.

A continuación, se define un concepto fundamental para las explicaciones basadas en lógica:

Definición 2 (Implicant y prime implicant)**.** Un término t se dice *implicant* de una fórmula φ si $t \models \varphi$. Asimismo, t es un *prime implicant* de φ si además no existe un subtérmino s de t tal que s sea un implicant de φ. Se nota $\texttt{PI}(\varphi)$ al conjunto de *prime implicants* de φ.

Ejemplo 2.1.5. Dadas las siguientes fórmulas booleanas:

- $\varphi = (x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3)$
- $\psi = (x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3)$

Notar que φ está en DNF, y ψ en CNF. Sus conjuntos de variables coinciden:

$$Var(\varphi) = Var(\psi) = \{x_1, x_2, x_3\}$$

Se puede comprobar que valen las siguientes afirmaciones:

- φ *no* es *monótona*, ya que los literales x_2 y $\overline{x_2}$ aparecen en la fórmula.

- ψ es *monótona*.
- $\varphi \models \psi$, pero $\psi \not\models \varphi$, pues la asignación $\{x_1 = 0, x_2 = 1, x_3 = 0\}$ satisface a ψ pero no a φ.
- φ y ψ *no* son *equivalentes* (por la afirmación anterior).
- El término $t = x_1 \wedge x_3$ es un *implicant* de φ.
- Además, t es un *prime implicant* de φ ya que ni x_1 ni x_3 son *implicants*.

La siguiente afirmación establece una forma de expresar una fórmula booleana en base a sus *prime implicants*:

Proposición 3. *Una fórmula booleana φ es equivalente a la disyunción de sus prime implicants:*

$$\varphi \equiv \bigvee_{t \in \mathit{PI}(\varphi)} t$$

Demostración. Sea $\mathtt{PI}(\varphi) = \{t_1, \ldots, t_k\}$ el conjunto de *prime implicants* de φ. Para probar que $\bigvee_{t \in \mathtt{PI}(\varphi)} t \models \varphi$, basta con notar que para todo i entre 1 y k, vale que $t_i \in \bigvee_{t \in \mathtt{PI}(\varphi)} t$ es un *implicant* de φ por definición. Luego, $t_i \models \varphi$. Entonces es evidente que al tomar la disyunción de los t_i, se preserva la implicación:

$$(t_1 \vee \ldots \vee t_k) \models \varphi$$

Para probar la otra dirección de implicación, considerar una asignación σ cualquiera que satisfaga a φ (i.e. $\varphi(\sigma) = 1$). Se construye ahora un término $m_\sigma = \bigwedge_{i=1}^{n} \ell_i$, donde:

$$\ell_i = \begin{cases} x_i & \text{si } \sigma(x_i) = 1 \\ \overline{x_i} & \text{si } \sigma(x_i) = 0 \end{cases}$$

Notar que este término evalúa a 1 bajo σ y además implica a φ. Luego, m_σ es un *implicant* de φ. Para obtener un *prime implicant*, se procede por pasos iterativos: remover un literal del término del paso actual, y dejarlo afuera si el término del paso siguiente sigue siendo un *implicant* de φ. Como la cantidad de literales es finita, el procedimiento finaliza con un término t_σ, al que no se le puede remover elementos sin que pierda su propiedad de *implicant*. Como σ era una asignación cualquiera, y se encontró un término t_σ tal que $t_\sigma(\sigma) = 1$ (pues solo contiene literales que σ satisface), entonces t_σ es un *prime implicant* de φ:

$$\varphi \models \bigvee_{t \in \mathtt{PI}(\varphi)} t$$

□

2.2. Clasificadores

En este trabajo, se centrará la atención en un tipo específico de problemas: los de clasificación. En el contexto del aprendizaje automático, los problemas de clasificación se definen sobre un conjunto de *features* (o atributos) $\mathcal{F} = \{x_1, \dots x_n\}$ y un conjunto de clases finito $\mathcal{K} = \{c_1, c_2, \dots, c_K\}$. Conceptualmente, las *features* toman valores formando la entrada del problema, y la clase representa el resultado de la clasificación.

Las *features* son elementos que forman un conjunto finito, mientras que su significado es definido por el contexto de uso. Cada *feature* $x \in \mathcal{F}$ toma valores de un dominio $\mathcal{D}_x = \{0, 1\}$. En otras palabras, se trabajará con *features* categóricas, y en particular booleanas. Sin embargo, la mayoría de resultados que se mostrarán a lo largo de esta tesis se pueden extender a dominios finitos arbitrarios, e incluso a *features* numéricas[2].

Dado un conjunto de *features* X, se pueden definir *entidades*. Una entidad es una asignación para cada $x \in X$. Se define el conjunto de entidades sobre X como

$$\texttt{ent}(X) = \{f : X \to \bigcup_{x \in X} \mathcal{D}_x \ : \ f(x) \in \mathcal{D}_x \ \forall\, x \in X\}$$

A partir de esta definición se puede ver que las entidades son el equivalente a las valuaciones en las fórmulas lógicas. Se usará la notación $e(x)$ para representar el valor asignado a la *feature* x en una entidad e. Además, en algunas oportunidades se hará un abuso de notación, usando una n-upla $(b_1, b_2, \dots, b_n)$ para representar una entidad, describiendo el valor que asigna a cada *feature*. Es decir, $b_1 = e(x_1), b_2 = e(x_2), \dots, b_n = e(x_n)$.

En cuanto al conjunto de clases $\mathcal{K}$, se trabajará con una cantidad finita k de ellas. En la mayoría de ocasiones se usarán dos clases (que se podrían interpretar como *verdadero* y *falso*) dado que es suficiente con ello para los resultados de complejidad que se demostrará. Un $k-$clasificador asigna una de las k clases a cada entidad; de esta forma, un clasificador $\mathcal{M}$ se puede caracterizar con una función de clasificación κ que toma valores en el espacio de asignaciones a las *features* (i.e. las entidades), y los envía al conjunto de clases $\mathcal{K}$, es decir $\kappa : \texttt{ent}(\mathcal{F}) \to \mathcal{K}$. Por lo tanto, es posible *evaluar* un clasificador $\mathcal{M}$ en una entidad e, lo cual ser notará como $\mathcal{M}(e) \in \mathcal{K}$. Se dice además que un $2-$clasificador (i.e. un clasificador booleano) *acepta* una entidad si responde positivamente para la misma ($\mathcal{M}(e) = 1$), y se dice que la *rechaza* si responde negativamente ($\mathcal{M}(e) = 0$). Es importante aclarar que en el resto de esta tesis, se supondrá que todo modelo (clasificador) puede ser evaluado en una asignación (valores para sus *features*) en tiempo polinomial en función de la descripción de dicho modelo.

La equivalencia de modelos se define en base a la evaluación: dados dos modelos $\mathcal{M}_1, \mathcal{M}_2$, se dice que $\mathcal{M}_1$ y $\mathcal{M}_2$ definidos sobre $\mathcal{F}$ son equivalentes (notado $\mathcal{M}_1 \equiv \mathcal{M}_2$) cuando $\mathcal{M}_1(e) = \mathcal{M}_2(e)$ para todas las entidades $e \in \texttt{ent}(\mathcal{F})$.

Se define el concepto de modelos constantes de la manera esperable:

Definición 4 (Modelo constante)**.** Un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$ se dice **constante** si responde de igual manera para cualquier entidad:

$$\exists\, b \in \{0, 1\} \mid \forall e \in \texttt{ent}(\mathcal{F}) : \mathcal{M}(e) = b$$

[2] Se profundiza en detalle acerca de esto en el artículo [17] publicado como *pre-impresión* en simultáneo con esta tesis.

Adicionalmente, se define el concepto de entidades *consistentes* con otra entidad en relación a un conjunto de *features*:

Definición 5 (Entidades consistentes)**.** Dado un conjunto de *features* X, y una entidad $e \in \texttt{ent}(X)$, se define el conjunto de *features* consistentes con e respecto de X (notado $\texttt{cw}(e, X)$, por *consistent with*) como aquel compuesto de las entidades que coinciden con e en todas las *features* de X:

$$\texttt{cw}(e, X) = \{e' \in \texttt{ent}(X) \,:\, (\forall x \in X \,:\, e'(x) = e(x))\}$$

Ejemplo 2.2.1. Sea $\mathcal{F} = \{x_1, x_2, x_3, x_4\}$ el espacio de *features*, $X = \{x_1, x_3\}$ un conjunto de *features* y $e = (1, 0, 0, 1)$. Entonces:

- $(1, 1, 0, 1) \in \texttt{cw}(e, X)$
- $(1, 0, 0, 0) \in \texttt{cw}(e, X)$
- $(1, 1, 1, 1) \notin \texttt{cw}(e, X)$
- $(0, 0, 1, 1) \notin \texttt{cw}(e, X)$

Se definen también las siguientes notaciones para simplificar la escritura, lectura y comprensión de la manipulación de modelos y entidades:

Definición 6 (Entidades y modelos con *feature* fijada)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \texttt{ent}(\mathcal{F})$, una *feature* $x \in \mathcal{F}$ y un valor $b \in \mathcal{D}_x$, se definen las siguientes notaciones:

- Se escribe $e_{x=b}$ a la única entidad que satisface: $e(y) = e_{x=b}(y)$ para toda *feature* $y \in \mathcal{F} \setminus \{x\}$, y $e_{x=b}(x) = b$.
- Se escribe $\mathcal{M}_{x=b}$ al modelo que satisface: $\mathcal{M}_{x=b}(e) = \mathcal{M}(e_{x=b})$ para $e \in \texttt{ent}(\mathcal{F})$.

Definición 7 (CNF y modelo restringidos a entidad)**.** Sea $\mathcal{M} \equiv \bigwedge_{i=1}^{m} c_i$ un modelo definido sobre un conjunto de *features* $\mathcal{F}$ y expresado como CNF, donde cada c_i es una cláusula definida sobre $\mathcal{F}$. Sea $L(\mathcal{F}) = \{x, \neg x \,|\, x \in \mathcal{F}\}$ el conjunto de literales sobre $\mathcal{F}$. Dada una entidad $e \in \texttt{ent}(\mathcal{F})$ y una cláusula $c = \bigvee_{j=1}^{m} \ell_j$, donde $\ell_j \in L(\mathcal{F})$, se define la cláusula c_i restringida a e como:

$$c_i^e := \bigvee_{\substack{\ell \in c_i \\ e \models \ell}} \ell$$

donde $e \models \ell$ si y solo si $(\ell = x \wedge e(x) = 1) \;\vee\; (\ell = \neg x \wedge e(x) = 0)$.

Luego, se define también el modelo $\mathcal{M}$ restringido a e como:

$$\mathcal{M}_e = \bigwedge_{i=1}^{m} c_i^e$$

En siguientes secciones se tratará con distintas familias de modelos, con diferentes grados de *succinctness* (i.e. qué tan *concisos* –o *sucintos*– son sus modelos miembros). Esto es porque, en algunos casos, los resultados y algoritmos que se propondrán solo aplican a ciertas familias de modelos. Además, existen ciertas propiedades que algunas familias de modelos cumplen, y otras no. Por ejemplo, la de estar cerradas por conjunción, disyunción, negación o condicionamiento:

Definición 8. Se dice que una familia $\mathcal{L}$ de modelos está (eficientemente) cerrada[3] por:

- *Conjunción* si para $\mathcal{M}_1, \mathcal{M}_2 \in \mathcal{L}$, vale que $\mathcal{M} = \mathcal{M}_1 \wedge \mathcal{M}_2 \in \mathcal{L}$, y además $\mathcal{M}$ se puede calcular eficientemente[4] dados $\mathcal{M}_1, \mathcal{M}_2$.
- *Disyunción exclusiva* si para $\mathcal{M}_1, \mathcal{M}_2 \in \mathcal{L}$, vale que $\mathcal{M} = (\mathcal{M}_1 \wedge x) \vee (\mathcal{M}_2 \wedge \overline{x}) \in \mathcal{L}$, donde x es una *feature* que no aparece en $\mathcal{M}_1$ ni $\mathcal{M}_2$; y además $\mathcal{M}$ se puede calcular eficientemente dados $\mathcal{M}_1, \mathcal{M}_2$.
- *Negación* si para $\mathcal{M} \in \mathcal{L}$, vale que $\neg\mathcal{M} = 1 - \mathcal{M} \in \mathcal{L}$, y además $\neg\mathcal{M}$ se puede calcular eficientemente dado $\mathcal{M}$.
- *Condicionamiento* si para $\mathcal{M} \in \mathcal{L}$, vale que $\mathcal{M}_{x=b} \in \mathcal{L}$ para toda $x \in \mathcal{F}$ y todo $b \in \mathcal{D}_x$; y además $\mathcal{M}_{x=b}$ se puede calcular eficientemente dado $\mathcal{M}$.

Para dar una visión más concreta de lo que representa un clasificador, se verán algunos ejemplos de clasificadores utilizados en la literatura [4, 9, 40], y que aparecen más adelante en esta tesis.

2.2.1. Árboles de decisión booleanos

El primer clasificador que se define es el más sencillo de los que aparecerán en este trabajo: la familia de los árboles de decisión booleanos.

Definición 9 (Árbol de decisión booleano)**.** Un árbol de decisión booleano T definido sobre un conjunto de *features* $\mathcal{F}$ es un árbol binario[5] donde:

- Cada vértice no terminal (o *interno*) de T está etiquetado con alguna de las variables de entrada (*features*) de $\mathcal{F}$.
- Cada vértice terminal (u *hoja*) está etiquetado con una constante booleana (1 o 0)[6].

Los *hijos* de cada vértice interno v (i.e. los vértices en el *vecindario –o vecindad– de salida* de v) se representan como un par ordenado (w_1, w_2); así, se puede hacer referencia a w_1 como el *hijo izquierdo* de v, y a w_2 como el *hijo derecho* de v (notados $\texttt{left}(v)$ y $\texttt{right}(v)$, respectivamente).

Se define el valor de T en una entidad de entrada e, notado $T(e) \in \{0, 1\}$ como el valor con el que está etiquetada la hoja alcanzada desde la raíz de la siguiente manera: en cada vértice, tomar la arista de la izquierda si la variable correspondiente a ese vértice en la entidad e es 0, y tomar la arista de la derecha en caso de que sea 1.

Dado un vértice no terminal $v \in T$, se define como $\texttt{feat}(v)$ a la *feature* con la cual está etiquetado v.

El tamaño de un árbol de decisión T se define como su cantidad de vértices, y se nota $|T|$.

[3] A partir de ahora, siempre que se haga referencia a una familia *cerrada* por alguna operación, se deja implícito que es *eficientemente cerrada*.

[4] Es decir, en tiempo polinomial respecto de (la suma del) tamaño de $\mathcal{M}_1$ y $\mathcal{M}_2$.

[5] En general, *booleano* se refiere a la cantidad de clases a las que puede asignar a una entrada, y binario a los valores que pueden tomar las *features*; sin embargo, se omitirán estas palabras cuando se deduzcan del contexto.

[6] Esto se puede extender a clases de más de dos elementos para el caso general (árboles de decisión binarios pero no necesariamente booleanos).

La clase de los árboles de decisión booleanos se denomina DT, por las siglas en inglés *Decision Tree*. En la Figura 2.1 se puede ver un pequeño ejemplo de un árbol de decisión booleano.[7]

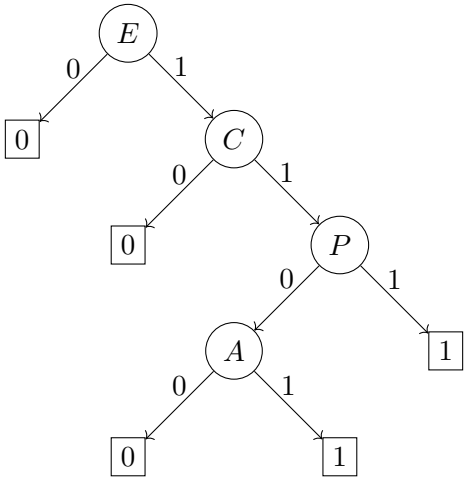


Fig. 2.1: Ejemplo de un árbol de decisión booleano que representa un clasificador para animales; este árbol acepta una entidad si ésta representa una tortuga, y la rechaza en caso contrario. El conjunto de *features* es E (tiene escamas), C (tiene caparazón), P (tiene cuatro patas), A (tiene cuatro aletas).

Sobre la relación entre árboles de decisión y fórmulas proposicionales, se puede asociar una fórmula en CNF a un árbol binario de decisión dado:

Definición 10 (CNF de un árbol de decisión)**.** Sea T un árbol de decisión booleano. Se define su conjunto de caminos desde la raíz hasta una hoja como $P_T = \{v_0, v_1, \ldots, v_k : v_0$ es la raíz de T, v_k es una hoja, y v_{i+1} es un hijo de v_i para $0 \leq i < k\}$. Se asociará un término booleano F_p al camino $p \in P_T$:

$$F_p \quad = \bigwedge_{\substack{0 \leq i < k \\ v_{i+1} = \texttt{left}(v_i)}} \overline{\texttt{feat}(v_i)} \quad \wedge \bigwedge_{\substack{0 \leq i < k \\ v_{i+1} = \texttt{right}(v_i)}} \texttt{feat}(v_i).$$

Se define ahora P_T^0 como el subconjunto de caminos que terminan en una hoja etiquetada con 0. Para cada $p \in P_T^0$, se define la cláusula

$$c_p := \neg F_p$$

, donde la negación se propaga según las leyes de De Morgan. Luego, la fórmula en CNF asociada a T se define como

$$\texttt{cnf}(T) = \bigwedge_{p \in P_T^0} c_p$$

Esta expresión es correcta pues para cada asignación $\mathbf{x}\colon \mathcal{F} \to \{0,1\}$ se tiene que $\mathbf{x}$ satisface a $\texttt{cnf}(T)$ si y solo si $\mathbf{x}$ no recorre ninguna rama que termine en 0 (y por lo tanto

[7] Este ejemplo se incluye meramente de modo ilustrativo, como una visualización de un árbol de decisión sencillo; no pretende ser un clasificador fiel o eficaz.

todas las ramas recorridas deben terminar en 1). Consecuentemente, $\texttt{cnf}(T) \equiv T$ en tanto funciones booleanas.

Además, el tamaño de la fórmula y el tiempo requerido para construirla son del orden de $O(|\mathcal{F}|\,|T|)$: la construcción de $\texttt{cnf}(T)$ consiste en recorrer todos los caminos desde la raíz hasta cada una de las hojas etiquetadas con 0 en T, y para cada uno de ellos generar una cláusula que niegue el término correspondiente al camino. El número de caminos desde la raíz a una hoja está acotado por la cantidad de hojas de T, que a su vez es menor o igual a la cantidad de vértices del árbol, i.e. $O(|T|)$. Para cada camino p de longitud ℓ, la construcción de su término asociado F_p requiere examinar cada uno de los ℓ vértices del camino. En el peor caso, $k = O(|\mathcal{F}|)$, ya que un árbol de decisión puede tener a lo sumo una *decisión* distinta por *feature* a lo largo de un camino. Luego, negar el término F_p según las leyes de De Morgan produce la cláusula c_p de tamaño $O(|\mathcal{F}|)$. Como hay $O(|T|)$ caminos, y cada cláusula correspondiente a un camino tiene longitud $O(|\mathcal{F}|)$, se concluye que la fórmula resultante tiene tamaño $O(|\mathcal{F}|\,|T|)$. El tiempo requerido para construir la fórmula es también $O(|\mathcal{F}|\,|T|)$, ya que cada cláusula se construye en tiempo proporcional a su tamaño, y se generan $O(|T|)$ cláusulas.

Esto significa que cualquier árbol de decisión puede ser representado por una fórmula en CNF, con un tamaño de orden polinomial respecto del tamaño del árbol. Contrariamente, no es posible representar cualquier fórmula en CNF como un árbol de decisión con similares garantías de tamaño. Sin embargo, en el siguiente resultado se muestra que cualquier modelo con una cantidad *pequeña* de entidades para las cuales responde positivamente, puede ser representado utilizando un árbol de decisión:

Lema 11 (De modelos pequeños a árboles de decisión)**.** *Sea $E \subseteq \texttt{ent}(X)$ un conjunto de entidades definidas sobre un conjunto de features X. Dado E, es posible construir un árbol de decisión binario T tal que $T(e) = 1$ si y solo si $e \in E$. Esta construcción se puede realizar en tiempo en el orden de $O(|E|\,|X|)$, y el árbol resultante tiene tamaño también en el orden de $O(|E|\,|X|)$.*

Demostración. Se construye el árbol de manera iterativa, empezando por un árbol que responda negativamente para todas las entidades (este árbol tiene un solo vértice etiquetado con 0).

Sea $E = \{e_1, \ldots, e_k\}$ el conjunto de entidades. Se supone además que, para algún $0 \leq i < k$, existe un árbol de decisión T_i que responde positivamente a las entidades $e_1, \ldots, e_i$, y rechaza el resto.

Dada la entidad e_{i+1}, se puede seguir el camino determinado en T para dicha entidad, y eventualmente se alcanzará una hoja etiquetada con 0 (por lo dicho antes). Tomar entonces este árbol, y extenderlo agregándole a esta hoja un árbol que responda positivamente solo para e_{i+1} (esto sería un camino de longitud $2\,|X|$).

El árbol resultante T_{i+1} responderá positivamente solo para las entidades $e_1, \ldots, e_i, e_{i+1}$, y tiene un tamaño menor o igual que $|T_i| + 2\,|X|$. □

2.2.2. Diagramas de decisión binarios

En un nivel de complejidad mayor al de los árboles de decisión binarios (en el sentido de, por ejemplo, la jerarquía de familias de modelos definida en [26]), se puede ubicar a los diagramas binarios de decisión [1] (o bien *binary decision diagrams*, BDD). Un BDD es una estructura de datos que representa una función booleana como un grafo acíclico

dirigido (o digrafo acíclico, o DAG). Cada vértice interno se corresponde con una decisión basada en una variable booleana (una *feature* booleana en este contexto), y cada arista representa una de las dos posibles respuestas a dicha decisión (verdadero o falso). Por su parte, las hojas del diagrama representan las constantes booleanas 1 y 0.

Existen familias de funciones $\{f_n\}_{n\in\mathbb{N}}$ (donde f_n tiene aridad n) cuya representación como árbol de decisión es exponencial en n (es decir, si T_n es el árbol de decisión que representa a f_n, vale que $|T_n| = \Omega(2^n)$), mientras que su representación como BDDs es compacta (i.e. polinomial en n). Es por este motivo que se dice que los BDDs son modelos con mayor *succinctness* que los árboles de decisión. Formalmente, se tiene que:

Definición 12 (BDD)**.** Un diagrama de decisión binario (BDD) definido sobre un conjunto finito de variables booleanas X es un grafo dirigido acíclico $G = (V, E)$ donde:

- Cada vértice interno $v \in V$ (i.e. tal que $d_{out}(v) > 0$) está etiquetado con una variable booleana $x \in X$, y cumple $d_{out}(v) = 2$.
- Cada arista $e = (v, w) \in E$ está etiquetada con un valor booleano (0 o 1), representando los posibles resultados de evaluar la variable x en el vértice v.
- Cada vértice terminal (u hoja) está etiquetado con una constante booleana (1 o 0).

A su vez, un BDD cumple las siguientes propiedades:

1. Ningún vértice tiene sus dos aristas salientes con la misma etiqueta.
2. El camino desde la raíz hasta alguna de las hojas determina inequívocamente un valor booleano (obtenido de manera equivalente a la mencionada para los árboles de decisión).
3. Sus subgrafos pueden ser compartidos entre distintos caminos, causando esto una mejora en eficiencia espacial respecto de una tabla de verdad.

Al igual que con los árboles de decisión, se nombrará $\texttt{right}(v)$ y $\texttt{left}(v)$ a los *hijos* derecho e izquierdo del vértice v, respectivamente; y se denota como $\texttt{feat}(v)$ a la *feature* asociada a v. En la Fig. 2.2, se muestra un ejemplo de BDD, que computa la función booleana $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$. El objetivo de este ejemplo es mostrar que los BDDs no necesariamente son árboles binarios, dado que el subgrafo que tiene como raíz a $\texttt{right}(x_3)$ comparte una rama con el subgrafo que tiene como raíz a $\texttt{left}(x_2)$[8].

Una clase especial de BDD son los FBDDs [29] (por las siglas de Free Binary Decision Diagrams, o *diagramas de decisión binarios libres*):

Definición 13 (FBDD)**.** Un diagrama de decisión binario libre (FBDD) es un BDD que cumple la *read-once property* (*propiedad de única lectura*): para todo camino desde la raíz hasta una hoja, cada variable puede aparecer a lo sumo una vez.

El nombre de la propiedad característica de los FBDDs proviene de que al momento de procesar uno de estos diagramas solamente se leerá (como máximo) una vez cada variable en cada uno de los caminos. Esto da lugar a mejoras en los algoritmos que operan sobre estas

[8] Estas expresiones ($\texttt{right}(x_3)$, $\texttt{left}(x_2)$) hacen un abuso de notación, dado que estos operadores se aplican sobre los vértices y no sobre sus variables asociadas. Sin embargo, como hay solo un vértice etiquetado con esas variables, se escribió así por simplicidad).

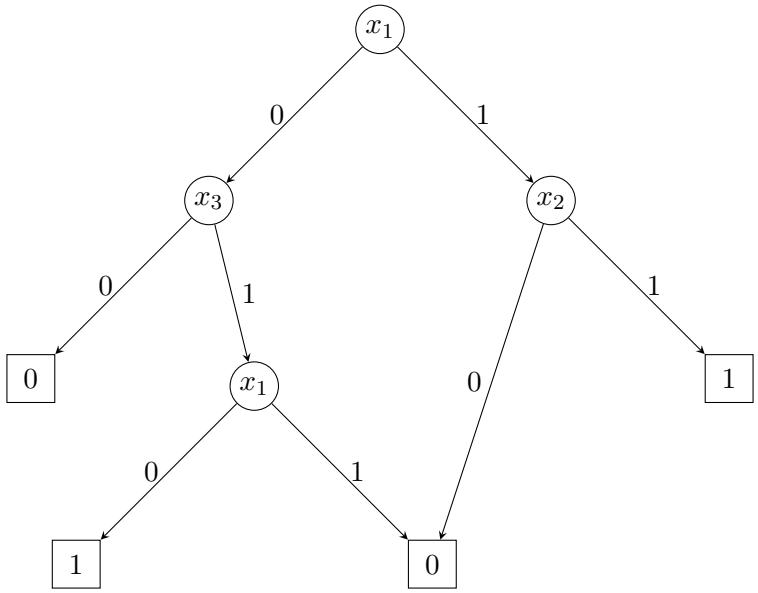


Fig. 2.2: BDD que representa la fórmula $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

estructuras, pues se puede aprovechar el hecho de que en el momento en el que el algoritmo procesó una variable en el camino, ésta nunca más va a ser procesada nuevamente.

En la Fig. 2.3 se puede ver un ejemplo de FBDD para la misma función que el BDD de la Fig. 2.2. Por un lado, notar que se cumple la *read-once property*, ya que los caminos desde la raíz hasta una hoja son:

- $x_1, x_3, 0$
- $x_1, x_2, 0$
- $x_1, x_3, x_2, 0$
- $x_1, x_2, x_3, 0$
- $x_1, x_3, x_2, 1$
- $x_1, x_2, x_3, 1$

Por otro lado, observar que el BDD de la Fig. 2.2 *no* cumple esta propiedad, dado que existe al menos un camino que repite a x_1 (e.g. $x_1, x_3, x_1, 1$); es por eso que ese diagrama no es un FBDD.

A su vez, una clase especial de FBDDs son los OBDDs [13] (diagramas de decisión binarios ordenados, u ordered decision binary diagrams):

Definición 14 (OBDD)**.** Un OBDD es un BDD que fuerza un orden en las variables (que en el caso de un clasificador, corresponden a las *features*) asociado a los vértices del diagrama. Para cada vértice v tal que $\texttt{feat}(v) = x_i$, y para $w \in \{\texttt{right}(v), \texttt{left}(v)\}$ tal que $\texttt{feat}(w) = x_j$ (siendo $\texttt{feat}(v)$ la *feature* asociada a v, y $\texttt{right}(v)$ y $\texttt{left}(v)$ los hijos derecho e izquierdo de v respectivamente, todo esto definido de manera análoga a lo hecho para FBDDs y árboles de decisión), debe valer $i < j$.

Esto significa que cualquier camino de la raíz a cada una de las hojas tiene que recorrer los vértices siguiendo un cierto orden: existe un ordenamiento $x_1 \prec x_2 \prec \ldots \prec x_n$ tal que todo camino $y_1, y_2, \ldots, y_k$ de la raíz a una hoja satisface $y_1 \prec y_2 \prec \ldots \prec y_k$. Notar que esta exigencia de orden también fuerza a que se cumpla la *read-once property* (pues si las variables deben aparecer en orden en cada camino desde la raíz hasta una hoja, entonces

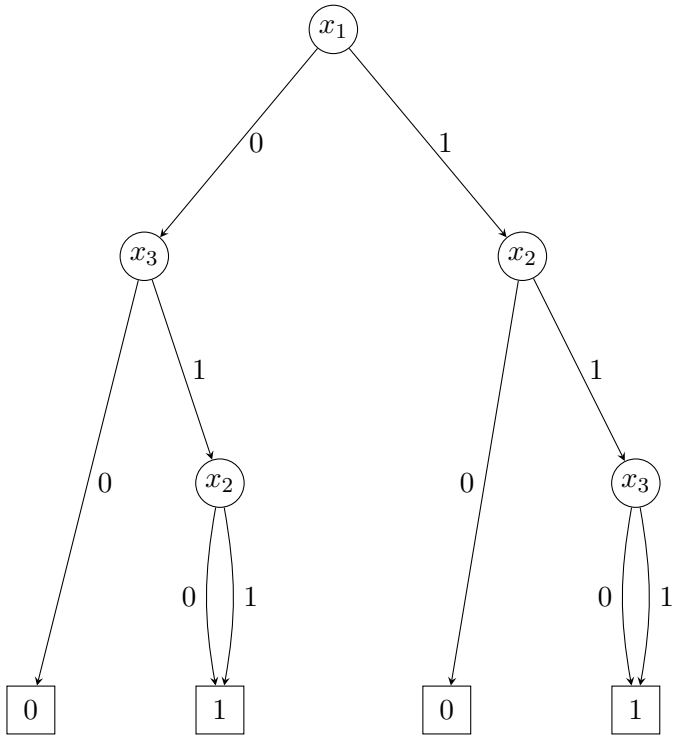


Fig. 2.3: FBDD que representa a la fórmula $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

en particular una variable no puede aparecer más de una vez en el camino). Por lo tanto, todo OBDD es también un FBDD (pero no vale la afirmación inversa).

En la Fig. 2.4 se puede ver un OBDD que calcula la misma fórmula que los diagramas anteriores. Se puede comprobar, por un lado, que éste es un FBDD (dado que se cumple la *read-once* property); pero además, existe un orden total en las variables de los vértices, que es $x_1 \prec x_2 \prec x_3$, con lo cual efectivamente es un OBDD, a diferencia del diagrama de la Fig. 2.3, que no fuerza un orden en las variables (en un camino x_2 aparece antes que x_3, y en otro camino ocurre a la inversa).

El requerimiento del orden total de los OBDDs puede ser aprovechado para diseñar algoritmos más eficientes que los admitidos por otros BDDs (incluso que los FBDDs). Sin embargo, no es la única ventaja que implica este orden: además, permite definir una forma canónica para la representación de OBDDs. Para eso, se define el concepto de OBDD *reducido*.

Definición 15 (Reducción de OBDDs)**.** Un OBDD se dice reducido cuando cumple simultáneamente las siguientes propiedades:

- Existe a lo sumo un vértice terminal (hoja) para una etiqueta dada (una hoja etiquetada con 1, y una con 0).
- No existe un vértice $v \in V$ tal que $\texttt{left}(v) = \texttt{right}(v)$
- No existen vértices no terminales $v, w \in V$ tales que $v \neq w$ y $\texttt{feat}(v) = \texttt{feat}(w)$, $\texttt{right}(v) = \texttt{right}(w)$ y $\texttt{left}(v) = \texttt{left}(w)$.

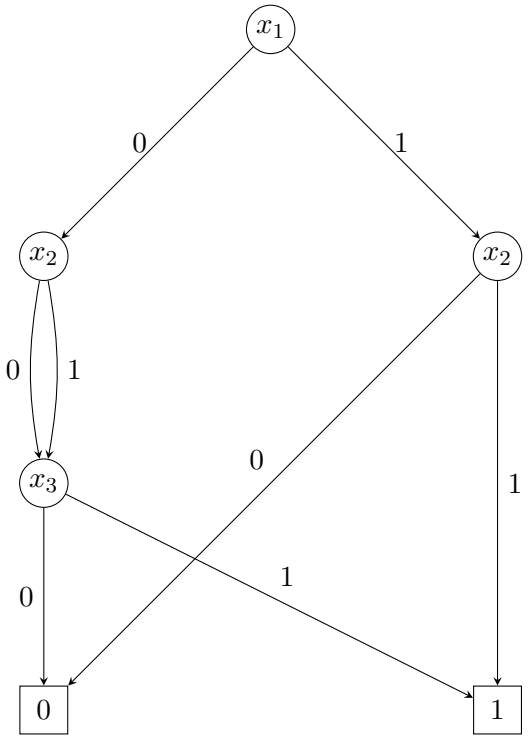


Fig. 2.4: OBDD que representa la fórmula $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$.

Esta caracterización permite dar con un procedimiento para convertir un OBDD cualquiera a su forma reducida, aplicando reiteradas veces las transformaciones que corresponden a las siguiente reglas [14]:

1. Si los vértices v y w son hojas y tienen la misma etiqueta, entonces remover una de ellas y redirigir todas sus aristas entrantes a la otra hoja.

2. Si el vértice v tiene $\texttt{right}(v) = \texttt{left}(v)$, entonces remover v y redirigir todas sus aristas entrantes a su hijo restante.

3. Si los vértices v y w tienen $\texttt{feat}(v) = \texttt{feat}(w)$, $\texttt{right}(v) = \texttt{right}(w)$ y $\texttt{left}(v) = \texttt{left}(w)$, entonces remover alguno de los dos y redirigir todas sus aristas entrantes al otro vértice.

Empezando desde las hojas del árbol, y subiendo por sus ramas, se puede completar esta reducción en tiempo lineal en el tamaño del grafo, como se menciona en [14]. De esta manera, se expone en [13] que un OBDD reducido es una forma canónica para funciones booleanas; es decir, fijando un orden de variables, toda función booleana sobre esas variables tiene una única representación como OBDD reducido (salvo isomorfismos). Esto permite verificar la equivalencia entre OBDDs de manera eficiente.

En la Fig. 2.5 se puede ver el OBDD reducido que computa la fórmula usada en los ejemplos anteriores.

Comentario. Respecto a las Fig. 2.2, 2.3 y 2.4: si bien es cierto que podrían tener menor cantidad de hojas (compartiendo más subgrafos), se decidió dibujarlos de esta manera para mayor claridad (evitando intersecciones entre aristas y curvas dentro de lo razonable). Por

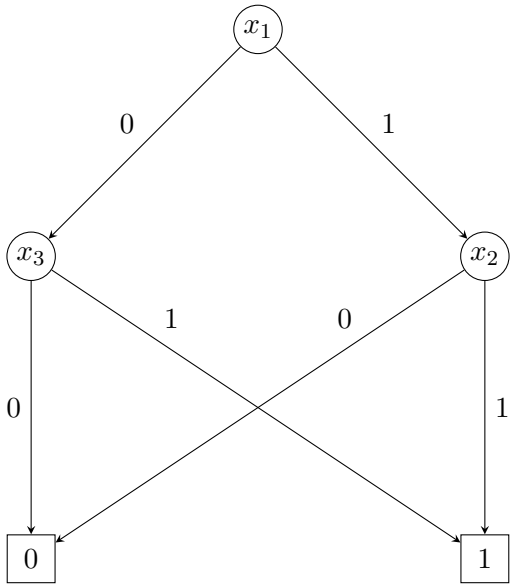


Fig. 2.5: Reducción del OBDD de la Fig. 2.4.

otro lado, notar que en la Fig. 2.2 no tiene sentido que se vuelva a consultar por x_1 después de x_3, dado que ya se había consultado eso mismo en la raíz del diagrama. Esto se hizo de esta manera por motivos meramente ilustrativos (i.e. dar un ejemplo de un BDD que no cumpla la *read-once property*). Lo mismo ocurre con los vértices de las Fig. 2.3 y 2.4 que tienen sus dos aristas salientes dirigidas al mismo vértice. Eso se puede simplificar eliminando el vértice innecesario, pero se decidió disponer los vértices de esta manera para proveer un balance entre la facilidad de lectura de los diagramas y la posibilidad de visualizar las propiedades que caraterizan a cada tipo de BDD.

2.2.3. Circuitos booleanos

Un circuito booleano es una estructura utilizada para representar la evaluación de funciones booleanas. Se trata de un digrafo acíclico que se caracteriza por el uso de compuertas lógicas, donde cada una de estas compuertas lleva a cabo una operación booleana en una o más entradas booleanas, para producir una salida binaria. Los componentes fundamentales de un circuito booleano son las variables de entrada (en el caso de un clasificador, las *features*), las compuertas lógicas, y la salida.

Formalmente, se puede definir a los circuitos booleanos la siguiente manera:

Definición 16. Un circuito booleano es una tupla $C = (V, E, g, I, O)$, donde:

- (V, E) es un grafo acíclico dirigido.
- $g : V \setminus (I \cup O) \to \{\texttt{AND}, \texttt{OR}, \texttt{NOT}\}$ es una función que asigna una compuerta lógica a cada vértice.
- $I \subseteq V$ es un conjunto de vértices de entrada (las variables de la función booleana). Además, $deg_{in}(v) = 0 \,\forall\, v \in I$.
- $O \subseteq V$ es un conjunto de vértices de salida (cada uno produce un resultado booleano). Además, $deg_{out}(v) = 0 \,\forall\, v \in O$

- $I \cap O = \emptyset$.
- Si $g(v) = \texttt{AND}$ o $g(v) = \texttt{OR}$, entonces $deg_{in}(v) \geq 2$, para todo $v \in Dom(g)$.
- Si $g(v) = \texttt{NOT}$, entonces $deg_{in}(v) \geq 1$, para todo $v \in Dom(g)$.

El proceso de cómputo (evaluación) en un circuito booleano comienza en los vértices de I, donde las variables booleanas se disponen como entrada. La evaluación procede a través de las compuertas lógicas, según la estructura definida por las aristas; eventualmente se llega a los vértices de salida en O. En la Fig. 2.6 se puede ver un circuito booleano que calcula la función $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$, al igual que los diagramas de las figuras anteriores.

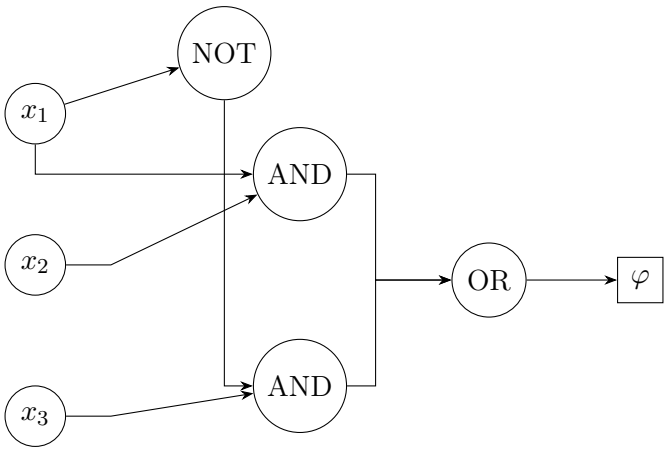


Fig. 2.6: Circuito booleano que representa la fórmula $\varphi(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

Comentario. En el contexto de este trabajo, siempre $|O| = 1$.

Existen diversas familias de clasificadores basados en circuitos booleanos, imponiendo restricciones adicionales para mejorar la tratabilidad de los problemas estudiados sobre ellos. Uno de los modelos que más aparece en la literatura de explicabilidad en IA es el de los circuitos d-DNNF [22] (decomposable and deterministic negated normal form). Este tipo de clasificadores está basado en la forma normal negada (NNF, *negated normal form*) y también se representa con digrafos acíclicos, permitiendo operadores de conjunción y disyunción con las siguientes restricciones:

- Decomposability: los caminos que descienden de una conjunción no comparten variables.
- Determinism: los caminos que descienden de una disyunción son inconsistentes entre sí (solo uno de los dos puede ser verdadero a la vez).

Los circuitos d-DNNF son una familia de clasificadores difundido en esta disciplina debido a que las propiedades que cumple le otorgan beneficios en cuanto a la tratabilidad de los problemas, y por otro lado posee un nivel de *succinctness* mayor al de los árboles de decisión o los OBDDs (sin llegar a ser tan potentes como los circuitos booleanos generales). Es decir, son un punto medio en el *trade-off* entre tratabilidad y *succinctness*. Además, existe investigación con resultados positivos acerca de la posibilidad de compilar modelos

más complejos a circuitos d-DNNF [23, 25, 60, 28], para luego aprovechar la tratabilidad de estos clasificadores para buscar explicaciones sobre las decisiones del modelo más complejo.

Por último, esta clase de modelos es la más *concisa* (con mayor *succinctness*) para la cual se puede calcular los *Shapley values* eficientemente, convirtiéndose ese problema en computacionalmente difícil si se elimina alguna de las dos restricciones (*decomposability* o *determinism*).

2.3. Explicabilidad basada en lógica

Antes de comenzar a discutir los problemas que conciernen a esta tesis, es necesario definir los conceptos básicos que hacen a la noción de *explicabilidad*. Intuitivamente, se puede decir que *algo* es *explicable* cuando es posible formular un *objeto* (el cual puede tomar distintas formas) que contribuya a que un determinado individuo consiga una mejor comprensión del mencionado *algo*.

En el caso de un modelo de inteligencia artificial, lo que debe ser explicado es el resultado al que arriba el modelo, para una cierta entidad de entrada. Haciendo foco en los problemas de clasificación, un modelo $\mathcal{M}$ asignará una categoría o clase c a una cierta entidad e. Para simplificar el escenario, se puede suponer que ese resultado es binario: *verdadero* (1) o *falso* (0). La tarea es, entonces, brindar una explicación *razonable* y *adecuada* para el resultado $\mathcal{M}(e) = c$. Antes de definir una explicación formalmente, se intentará responder tres preguntas que surgen de este razonamiento:

- ¿Qué significa que una explicación sea *razonable* o *adecuada*? Una manera de definir estos conceptos aplicados a explicaciones en inteligencia artificial es la siguiente: depende del grupo de personas a quien esté dirigida la explicación. Es decir, una explicación será *razonable* si la persona que la recibe puede entenderla de alguna manera, y *adecuada* si el hecho de entenderla le ayuda en la comprensión de su situación (es decir, de la parte de su realidad que se está viendo afectada por la decisión del modelo). Por ejemplo, el modelo entero es una *explicación* en el sentido de que allí se puede encontrar la justificación del comportamiento del mismo. Sin embargo, esto no cumple con las características de lo que aquí se considera una *explicación razonable*: para empezar, es cognitivamente inabordable para un individuo, y el exceso en el nivel de detalle oculta la intuición que le podría ser útil al destinatario. Entonces, una *explicación* funciona como una *compresión* de la realidad: esta compresión debe ser lo suficientemente detallada como para que pueda *justificar* una decisión, pero a la vez debería ser concisa, a través de eliminar detalles que no resulten importantes para el destinatario.

- ¿A quién está dirigida la explicación? Este destinatario podría ser el grupo directamente afectado por la decisión del modelo, como en el caso de un cliente solicitando un crédito; o también podría ser una persona con un cierto conocimiento técnico elevado del campo, que luego podría comunicarle la explicación a otra persona menos experta, como por ejemplo un médico explicando el diagnóstico asistido por inteligencia artificial a su paciente. Por lo tanto, no es tan importante la forma que tenga la explicación, mientras esta pueda ser correctamente interpretada por quien corresponda.

- ¿La explicación tiene algún sustento? Esto es, si existe o no un marco teórico formal que provea algún tipo de garantía acerca de la corrección de la explicación. Esto es importante sobre todo en contextos de uso especialmente sensibles, como los mencionados en la introducción de esta tesis.

Existen diversos enfoques a la hora de construir y dar forma a las explicaciones. Aquí se centrará la atención principalmente en el *feature selection* basado en lógica, es decir seleccionar las *features* de un modelo que cumplan con ciertas propiedades lógicas. Para esto, en diversos trabajos se ha utilizado nociones como *logic abduction* [30, 61] o *prime*

implicant para reunir conjuntos de *features* que, con ciertos valores asignados, determinan el resultado para una cierta entidad y modelo. Por otro lado, el principal problema de la explicabilidad basada en lógica es la complejidad computacional que acarrean los procedimientos de búsqueda de explicaciones. Como se verá más adelante, no son demasiados los clasificadores para los cuales existen, a la fecha, buenos resultados de complejidad en el cálculo de explicaciones, sobre todo en cuanto a la relevancia. Sin embargo, en el transcurso de este trabajo aparecerán varios escenarios en los cuales las tareas de explicabilidad son tratables.

Dicho esto, es natural preguntarse cómo responden las explicaciones formales que se presentan en este trabajo (conjuntos de *features* que *determinan lógicamente* el resultado para una entrada) a las preguntas planteadas previamente. En cuanto a las primeras dos preguntas, dada la naturaleza teórica de estas explicaciones, su destinatario más inmediato sería alguien que tenga conocimiento acerca de las *features* sobre las cuales está definido un modelo dado, ya que una persona cualquiera no tiene por qué conocer estas *features*, ni tampoco los conceptos de relevancia, necesidad o sufficient reason. Si una de estas explicaciones debe ser transmitida a un usuario promedio de un sistema de inteligencia artificial, sería necesario un paso intermedio de procesamiento, con el objetivo de poder otorgar una justificación fácil de entender (por ejemplo, sintetizando las *features* seleccionadas en forma de una oración en lenguaje natural). Respecto a qué tan *buena* es la compresión de la realidad que produce una explicación de este tipo, se puede decir que debería cuanto menos *concisa* en algún sentido, puesto que la propuesta es estructurar una explicación como cierto conjunto minimal de *features* que determine el resultado. Por otro lado, la respuesta a la última pregunta es definitivamente afirmativa: justamente el principal atractivo de la explicabilidad basada en lógica es su sustento formal en conceptos matemáticos sólidos. Es en esto que este enfoque en el campo de XAI tiene una ventaja respecto de otros acercamientos como pueden ser los basados en aproximaciones numéricas a los *Shapley values*, como por ejemplo el muy difundido SHAP score [50].

2.3.1. Explicaciones

A continuación se definirá formalmente los dos tipos principales de explicación basada en lógica. Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$, y una entidad $e \in \mathtt{ent}(\mathcal{F})$. Sea $X \subseteq \mathcal{F}$ un subconjunto de las *features* de $\mathcal{M}$:

Definición 17 (Abductive explanation)**.** Se dice que X es una abductive explanation (o *explicación abductiva*) para $\mathcal{M}(e)$ si vale $AX(X)$, donde:

$$AX(X) := \forall\, e' \in \mathtt{ent}(\mathcal{F}) : \big(\bigwedge_{x \in X} (e'(x) = e(x)) \big) \rightarrow (\mathcal{M}(e') = \mathcal{M}(e))$$

Definición 18 (Contrastive explanation)**.** Se dice que X es una contrastive explanation (o *explicación contrastiva*) para $\mathcal{M}(e)$ si vale $CX(X)$, donde::

$$CX(X) := \exists\, e' \in \mathtt{ent}(\mathcal{F}) : \big(\bigwedge_{x \notin X} (e'(x) = e(x)) \big) \wedge (\mathcal{M}(e') \neq \mathcal{M}(e))$$

Las abductive explanations[9] representan los *prime implicants* de la función de clasificación de un modelo clasificador. También se pueden ver estas explicaciones como una

[9] Este es el nombre que se optó por darle a este concepto aquí; existen otros trabajos en los que se nombra a estas explicaciones como *weak* abductive explanations, o como PI-explanations.

forma de *logic-based abduction* (de ahí su nombre). La idea es que, dado un modelo $\mathcal{M}$, una entidad e y una clase c tales que $\mathcal{M}(e) = c$, una abductive explanation para $\mathcal{M}(e)$ es un subconjunto de *features* que cumple que: si se fija el valor de cada *feature* al valor determinado por la entidad e, entonces el resultado de evaluar $\mathcal{M}$ en una entidad resultante será c sin importar el valor asignado al resto de *features*.

Intuitivamente, son las *features* que determinan una clasificación; aquellas que, fijadas en valores dados por la entidad, e independientemente de las demás *features*, fuerzan el resultado provisto por el modelo. Se puede decir que son las explicaciones que muestran *por qué* una entidad se clasificó de cierta manera (en este caso, con la clase c).

Por otro lado, una contrastive explanation es, dado un modelo $\mathcal{M}$, una entidad e y una clase c tales que $\mathcal{M}(e) = c$, un subconjunto de *features* tal que, si se permite que cada una tome cualquier valor de su dominio (en el caso que se analiza aquí, sería 0 o 1), entonces existe una asignación de las *features* que cambia la predicción a una clase diferente c', todo esto mientras las *features* que no pertenecen a la contrastive explanation se mantienen con el valor dictado por la entidad e.

Haciendo una interpretación como la dada a las abductive explanations, se puede decir que una contrastive explanation es una posible respuesta a la pregunta de *¿por qué no?*, en el sentido de por qué el clasificador no respondió con una clase que no sea c. Otra manera de pensarlo es como la respuesta a una pregunta de *cómo* cambiar el valor de las *features* para que varíe el resultado. Además, las contrastive explanations son particularmente útiles cuando una entidad no es clasificada por el modelo como uno habría esperado; en este caso, uno puede buscar los subconjuntos de *features* que, cuando cambian su valor asignado en la entidad, resultan suficientes para conseguir otras entidades que sí sean clasificadas de la manera en la que se esperaba que lo fuera aquella entidad original. No sólo eso, sino que las contrastive explanations también pueden ser útiles en el escenario opuesto: cuando una entidad fue clasificada efectivamente como uno lo esperaba; en estos casos, las contrastive explanations informan de manera precisa cómo variar de manera minimal las entidades para conseguir un resultado diferente. Esto puede ayudar a evaluar la robustez de la clasificación provista por el modelo.

En esta tesis, se centrará la atención en las abductive explanations, y específicamente en una subclase de estas explicaciones: las minimales. Algunos trabajos que profundizan más sobre las contrastive explanations son [48, 45, 6].

2.3.2. Sufficient reasons

Se define como sufficient reasons[10] a las abductive explanations *minimales*: es decir, aquellas abductive explanations tales que, si se les remueve cualquiera de sus elementos, dejarían de ser abductive explanations.

Definición 19 (Sufficient reason)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \mathtt{ent}(\mathcal{F})$, y un subconjunto de *features* $X \subseteq \mathcal{F}$, se dice que X es una sufficient reason (o *razón suficiente*) para $\mathcal{M}(e)$ (expresado también como "para e dado $\mathcal{M}$") si vale $SR(X)$, donde:

$$SR(X) := AX(X) \ \wedge \ (\forall Y \subset X : \neg AX(Y))$$

[10] En otros trabajos donde las abductive explanations se nombran *weak* abductive explanations, se conoce simplemente como abductive explanation a lo que aquí se define como sufficient reason. Es posible encontrar también a las sufficient reasons definidas como *minimal* abductive explanations.

Son aquellos conjuntos de *features* que son *suficientes* para determinar el resultado que dará el modelo para una entidad; además, es por esta definición que a veces se conoce como *reasons* a las abductive explanations que no son minimales.

Se notará como $SR(\mathcal{M}, e)$ al conjunto de todas las sufficient reasons para la entidad e dado el modelo $\mathcal{M}$.

Ejemplo 2.3.1. Sea $\mathcal{M}$ un modelo definido sobre $\mathcal{F} = \{x_1, x_2, x_3, x_4, x_5\}$, y la entidad $e = (1, 1, 1, 1, 1)$. La función computada por $\mathcal{M}$ es $\varphi(x_1, x_2, x_3, x_4, x_5) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$, con lo cual notar que $\mathcal{M}(e) = 1$.

Las siguientes son abductive explanations para $\mathcal{M}(e)$:

- $X_1 = \{x_1, x_2\}$ cumple $AX(X_1)$ ya que fijando $x_1 := e(x_1) = 1$ y $x_2 := e(x_2) = 1$ se tiene que la primera conjunción $(x_1 \wedge x_2)$ es verdadera, con lo cual $\mathcal{M}(e') = \mathcal{M}(e) = 1$ para toda entidad e' que coincida con e en X_1.
- $X_2 = \{x_1, x_3, x_4\}$ cumple $AX(X_2)$ pues cualquier entidad e' que fije $x_1 := e(x_1) = 1$, $x_3 := e(x_3) = 1$ y $x_4 := e(x_4) = 1$, cumplirá con la segunda conjunción $(x_3 \wedge x_4)$, con lo cual $\mathcal{M}(e') = \mathcal{M}(e) = 1$ para toda entidad e' que coincida con e en X_2.

X_1 es además una sufficient reason para e dado $\mathcal{M}$, ya que si se remueve x_1 o x_2 se rompe la validez de la primera conjunción de φ; es decir que es minimal. Sin embargo, X_2 no es minimal ya que $X_2' = \{x_3, x_4\}$ también cumple con la definición de abductive explanation. Entonces, X_2 no es sufficient reason pero X_2' sí.

2.3.3. Features relevantes, irrelevantes y necesarias

Existe un concepto muy relacionado con las sufficient reasons y las explicaciones en general[11]: la relevancia y necesidad de *features* . Se trata de analizar la *importancia* de las *features* de un modelo para una cierta entidad. Es decir, el estudio de qué *features* aparecen en alguna sufficient reason, o en todas, o en ninguna. Estos conceptos son útiles y tiene sentido estudiarlos, ya que aquellas *features* que no aparezcan en las explicaciones para una clasificación del modelo, serán intuitivamente las que menor peso tengan a la hora de explicar un resultado.

A continuación, se define la primera categoría de *features* de interés:

Definición 20 (Feature relevante)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$ y una entidad $e \in \texttt{ent}(\mathcal{F})$, se dice que una *feature* $x \in \mathcal{F}$ es *relevante* para $\mathcal{M}(e)$ (también expresado como "relevante para e dado $\mathcal{M}$") cuando x pertenece a al menos una sufficient reason para $\mathcal{M}(e)$:

$$Rel(x, \mathcal{M}, e) := \exists\, X \in SR(\mathcal{M}, e) \,|\, x \in X$$

Se define asimismo y por completitud la categoría complementaria a las *features* relevantes, que no será explorada en detalle durante este trabajo:

Definición 21 (Feature irrelevante)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$ y una entidad $e \in \texttt{ent}(\mathcal{F})$, se dice que una *feature* $x \in \mathcal{F}$ es *irrelevante* para $\mathcal{M}(e)$ (también expresado como "irrelevante para e dado $\mathcal{M}$") cuando x no pertenece a ninguna sufficient reason para $\mathcal{M}(e)$:

$$Irrel(x, \mathcal{M}, e) := \forall\, X \in SR(\mathcal{M}, e) \,:\, x \notin X$$

[11] Se puede definir para otros tipos de explicaciones, como las contrastive explanations.

La siguiente categoría es la de aquellas *features* que aparecen en todas las sufficient reasons:

Definición 22 (Feature necesaria)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$ y una entidad $e \in \texttt{ent}(\mathcal{F})$, se dice que una *feature* $x \in \mathcal{F}$ es *necesaria* para $\mathcal{M}(e)$ (también expresado como "necesaria para e dado $\mathcal{M}$") cuando x pertenece a todas las sufficient reasons para $\mathcal{M}(e)$:

$$Nec(x, \mathcal{M}, e) := \forall\, X \in SR(\mathcal{M}, e) \,:\, x \in X$$

Observación 23. *Si una feature es necesaria para una entidad e dado un modelo $\mathcal{M}$ (y además existe al menos una sufficient reason no vacía*[12]*) entonces también es relevante.*

Ejemplo 2.3.2. Dados el modelo $\mathcal{M}$ y la entidad e del Ejemplo 2.3.1:

- Las *features* relevantes son x_1, x_2, x_3, x_4.
- La única *feature* irrelevante es x_5.
- No hay *features* necesarias.

Si se alterase la fórmula φ computada por el modelo $\mathcal{M}$ de la siguiente manera: $\psi \equiv x_1 \wedge \varphi$, entonces las *features* relevantes e irrelevantes serían las mismas, pero x_1 sería necesaria.

Intuitivamente, las *features* necesarias (resp. irrelevantes) para una entidad e y un modelo $\mathcal{M}$ son las más (resp. menos) importantes para explicar la clasificación de e por parte de $\mathcal{M}$, dado que pertenecen a toda (resp. ninguna) sufficient reason para e dado $\mathcal{M}$.

Por otro lado, es posible desarrollar métodos de *feature ranking* basados en la noción de relevancia de *features*: por ejemplo, calculando una cierta puntuación para cada *feature* dependiendo de a cuántas explicaciones pertenezca, y ordenando las *features* del modelo.

Es evidente la relación existente entre las *features* relevantes y las sufficient reasons: una sufficient reason sólo contiene *features* relevantes, y las *features* relevantes son aquellas que pertenecen a alguna sufficient reason. Es claro también que si fuese posible calcular todas las sufficient reasons, entonces se podría saber fácilmente cuáles son las *features* relevantes. Contrariamente, no es tan simple obtener todas las sufficient reasons a partir de una enumeración de las *features* relevantes. Pero ambos conceptos son útiles para contribuir a la explicabilidad de los modelos de inteligencia artificial (o de cualquier modelo que cumpla las definiciones y propiedades vistas previamente), y por lo tanto serán el objeto de estudio de este trabajo.

2.3.4. Features útiles

Además de las *features* relevantes, irrelevantes y necesarias, se definirá ahora un concepto que surgió mientras se investigaba acerca de los algoritmos para decidir relevancia de features; se trata de las aquí nombradas *features útiles*.

Definición 24 (Feature útil)**.** Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$, se dice que una *feature* $x \in \mathcal{F}$ es *útil* para $\mathcal{M}$ si existe una entidad $e \in \texttt{ent}(\mathcal{F})$ y un valor $b \in \mathcal{D}_x$ tales que $\mathcal{M}(e) \neq \mathcal{M}(e_{x=b})$.

[12] Esta condición es técnica pero necesaria, dado que si el conjunto de sufficient reasons fuese vacío, entonces trivialmente todas las *features* serían necesarias por definición; sin embargo, no serían relevantes.

Intuitivamente, una *feature* no es útil cuando el modelo no la requiere, ya que no existe ninguna entidad cuya clasificación dependa del valor de dicha *feature*. Esta intuición se comprueba formalmente, dado que si x no es útil para $\mathcal{M}$, entonces $\mathcal{M} \equiv \mathcal{M}_{x=b}$ para cualquier valor $b \in \mathcal{D}_x$.

Notar que, a diferencia de lo que sucede para las *features* relevantes, irrelevantes y necesarias, las *features* útiles no dependen de una entidad, sino que solamente dependen del modelo. Es por esto que, desde un punto de vista conceptual, se puede afirmar que la noción de *utilidad* es una cualidad de una *feature* referente al modelo en general, y no a una predicción o resultado en particular. La idea es justamente capturar una noción de relevancia global de una *feature* para un modelo.

2.4. Problemas y resultados adicionales

Por último, se introducen algunos conceptos que no están directamente relacionados con la explicabilidad, pero que serán de utilidad para el estudio de los problemas que ocupan la atención de esta tesis.

Satisfacibilidad en fórmulas CNF monótonas Se define una variante del problema de decidir si una fórmula proposicional es satisfacible, que será útil para otro problema que se estudiará más adelante. Se trata de una versión del problema en la que la fórmula (llamada ϕ) es monótona, y es una conjunción de dos fórmulas más pequeñas (ϕ^+ y ϕ^-), que cumplen ciertas características sobre la polaridad de sus variables. Considerar el problema definido como:

> PROBLEMA: 3,2-CNF-MONOTONE-SAT
> ENTRADA: Una fórmula booleana $\phi = \phi^+ \wedge \phi^-$, donde ϕ^+ es una fórmula monótona en 3-CNF con todas sus variables no negadas, y ϕ^- es una fórmula monótona en 2-CNF con todas sus variables negadas.
> SALIDA: Existe una asignación $\sigma : Var(\phi) \to \{0, 1\}$ para las variables de ϕ que satisface a ϕ (i.e. $\phi(\sigma) = 1$).

Ejemplo 2.4.1. Considerar las siguientes fórmulas booleanas:

- $\varphi = ((x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4)) \wedge ((\neg x_1 \vee \neg x_2) \wedge (\neg x_3 \vee \neg x_4))$
- $\psi = ((x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3))$
  $\wedge ((\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3))$

La parte de φ (resp. ψ) en color verde corresponde a φ^+ (resp. ψ^+), mientras que la parte violeta corresponde a φ^- (resp. ψ^-).

La asignación $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$ evalúa a 1 en φ, con lo cual para esta instancia 3,2-CNF-MONOTONE-SAT responde positivamente.

Cada cláusula de ψ^+ omite exactamente una variable, con lo cual para satisfacer todas sus cláusulas son necesarias al menos dos variables con valor 1, ya que si se tiene una sola variable en 1, fallará para la cláusula que no la incluya. Por otra parte, ψ^- fuerza a que no pueda haber ningún par de variables con valor 1 simultáneamente. En consecuencia, ψ no es satisfacible, con lo cual 3,2-CNF-MONOTONE-SAT responde negativamente.

Hitting sets Dado un conjunto finito X de elementos cualesquiera, y m subconjuntos $B_1, \ldots, B_m \subseteq X$, se conoce como hitting set a un conjunto $S \subseteq X$ que contiene al menos un elemento de todo B_i para i entre 1 y m (se dice que S *golpea* –hace *hit*– a todos estos conjuntos). A su vez, un hitting set se dice minimal cuando al remover cualquiera de sus elementos deja de ser un hitting set.

Ejemplo 2.4.2. Dado $X = \{x_1, x_2, x_3, x_4\}, B_1 = \{x_2, x_3\}$ y $B_2 = \{x_4\}$, el conjunto $S = \{x_2, x_3, x_4\}$ es un hitting set, pero no es minimal: al remover x_3, el resultado $S' = \{x_2, x_4\}$ es también un hitting set (que de hecho sí es minimal).

En este trabajo será de interés una variante del problema de decidir si existe un hitting set: se trata de introducir la noción de *pares prohibidos*: se exige que ciertos pares de elementos no puedan estar incluidos en el hitting set. Esto induce el siguiente problema de decisión:

PROBLEMA: HITTING-SET-FP
ENTRADA: Un conjunto X; m conjuntos $B_1, \ldots, B_m \subseteq X$; k pares de elementos $p_1, \ldots, p_k$ tales que $p_i = (y_{i,1}, y_{i,2})$, con $y_{i,1}, y_{i,2} \in X$ para todo i, con $1 \leq i \leq k$.
SALIDA: Existe un hitting set $S \subseteq X$ para $B_1, \ldots, B_m$ (i.e. conjunto que contiene al menos un elemento de cada $B_1, \ldots, B_m$) tal que ningún par $p_1, \ldots, p_k$ esté contenido en S.

Es decir, se responde afirmativamente si existe un subconjunto de X que cubra a todos los conjuntos B_i, y que no contenga a ciertos pares de elementos simultáneamente. Esto se puede nombrar como el problema de *hitting set con pares prohibidos*.

Intuitivamente, se puede pensar el problema como tomar (al menos) un elemento de cada *bolsa* B_i para meter en la bolsa de la solución S.

Ejemplo 2.4.3. Considerar la siguiente instacia para HITTING-SET-FP:

$$X = \{x_1, x_2, x_3, x_4\}, B_1 = \{x_1, x_2\}, B_2 = \{x_3, x_4\}, p_1 = (x_1, x_3)$$

Un conjunto que cumple todas las condiciones pedidas es $S = \{x_1, x_4\}$, pues:

- S contiene a un elemento de B_1 (x_1) y a uno de B_2 (x_4).
- S no contiene a los dos elementos del *par prohibido* p_1 (contiene a x_1 pero no a x_3).

Entonces, HITTING-SET-FP responde positivamente para esta instancia. Considerar ahora otra instancia para HITTING-SET-FP:

$$X = \{x_1, x_2\}, B_1 = \{x_1\}, B_2 = \{x_2\}, p_1 = (x_1, x_2)$$

Se puede ver que es una instancia en la que el problema responde negativamente (S debería ser $\{x_1, x_2\}$ pero eso viola el par prohibido p_1).

Vertex covers Un vertex cover (o cubrimiento de aristas por vértices) es un conjunto de vértices pertenecientes a un grafo, tal que todas las aristas del grafo están *cubiertas* por alguno de los vértices:

Definición 25 (Vertex cover)**.** Dado un grafo $G = (V, E)$, un cubrimiento de aristas por vértices (o vertex cover en inglés) es un conjunto de vértices $\mathcal{C} \subseteq V$ tal que:

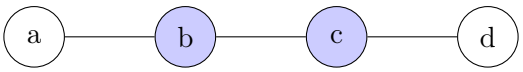
$$(v, w) \in E \implies v \in \mathcal{C} \vee w \in \mathcal{C}$$

Un vertex cover se dice *minimal* cuando el resultado de quitarle un vértice cualquiera es un conjunto que no cumple la definición de vertex cover.

Notar que el problema de encontrar un vertex cover minimal no es demasiado interesante, dado que siempre existe uno (es computacionalmente *fácil* de resolver). Sin embargo, en esta tesis se considera una variante de dicho problema, que agrega una restricción adicional: todos los vértices de un conjunto W deben estar contenidos en el vertex cover:

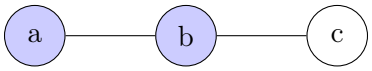
PROBLEMA: RESTRICTED-VERTEX-COVER
ENTRADA: Un grafo $G = (V, E)$, un conjunto de vértices $W \subseteq V$.
SALIDA: Existe un vertex cover minimal $\mathcal{C} \subseteq V$ para G, tal que $W \subseteq \mathcal{C}$.

Ejemplo 2.4.4. Considerar la siguiente instancia de RESTRICTED-VERTEX-COVER, dada por el grafo G que se muestra a continuación, y el conjunto $W = \{b, c\}$:



En este caso, la respuesta sería positiva, dado que el conjunto $\mathcal{C} = \{b, c\}$ es un vertex cover de G que contiene a W, y es minimal.

Por otro lado, la siguiente instancia, dada por el grafo G dibujado abajo y $W = \{a, b\}$, es negativa, ya que no existe un vertex cover que incluya a a y b y además sea minimal (para que sea minimal, habría que remover a, lo cual violaría la restricción de contener a W):



Transversals en hipergrafos Un hipergrafo es una generalización del concepto de grafo, en el que una (hiper-)arista puede conectar cualquier número de vértices, y no solo dos. Formalmente, es un par $H = (V, E)$, donde:

- V es un conjunto de vértices.
- E es un conjunto de *hiperaristas*.

Cada hiperarista tiene la forma $B \subseteq V$ (las hiperaristas son conjuntos de vértices de H).

Entonces, se puede decir que una hiperarista establece una relación entre conjuntos de vértices del hipergrafo. Existen a su vez los equivalentes a conceptos propios de grafos en los hipergrafos; por ejemplo, el grado de un vértice $v \in V(H)$ se define como la cantidad de hiperaristas a las que pertenece v:

$$deg(v) := |\{B \in E : \ v \in B\}|$$

El tamaño de un hipergrafo H se nota $|H|$, y se define como:

$$|H| := |V| + \sum_{B \in E} |B|$$

Los problemas clásicos para grafos tienen su versión para hipergrafos. Uno de ellos es el del vertex cover minimal; el vertex cover (cubrimiento de aristas por vértices) se suele conocer como *hypegraph transversal*, y se refiere a un conjunto de vértices $\mathcal{C}$ tal que cada hiperarista del hipergrafo contenga a al menos un vértice $\mathcal{C}$.

Definición 26 (Hypergraph transversal)**.** Dado un hipergrafo $H = (V, E)$, un *transversal* (o vertex cover) es un conjunto de vértices $\mathcal{C} \subseteq V$ tal que:

$$\forall B \in E : \ \mathcal{C} \cap B \neq \emptyset$$

Un transversal $\mathcal{C}$ es minimal si para todo $\mathcal{C}' \subset \mathcal{C}$, se tiene que $\mathcal{C}'$ no es un transversal.

Se define también una variante del problema del transversal minimal en hipergrafos, en el que, al igual que se hizo con el caso de los vertex covers, se requiere que el transversal contenga a todos los vértices de un conjunto:

PROBLEMA: RESTRICTED-TRANSVERSAL ENTRADA: Un hipergrafo $H = (V, E)$, un conjunto de vértices $W \subseteq V$. SALIDA: Existe un transversal minimal $\mathcal{C} \subseteq V$ para H, tal que $W \subseteq \mathcal{C}$.

Notar que el problema de hallar un transversal para un hipergrafo es equivalente al de encontrar un hitting set para un conjunto. En el primer caso, se busca un conjunto de vértices que incluya al menos un vértice de cada hiperarista, y en el segundo se busca un conjunto de elementos que incluya al menos un elemento de cada conjunto. La diferencia principal este HITTING-SET-FP y RESTRICTED-TRANSVERSAL radica en que en HITTING-SET-FP se requiere que ciertos pares de elementos no estén incluidos en el conjunto solución; en cambio, en el caso de RESTRICTED-TRANSVERSAL se requiere que ciertos elementos sí estén incluidos en la solución (además de exigirse minimalidad en este último).

Ahora se enunciarán algunos resultados que, si bien no son definiciones, sirven como conceptos preliminares para el análisis de complejidad computacional de los problemas que se estudiarán en esta tesis.

Existe una relación de dualidad entre las sufficient reasons y los transversals minimales en hipergrafos (o equivalentemente, hitting sets minimales)[47, 12, 62]:

Lema 27 (Sufficient reasons y transversals minimales)**.** *Sea $\mathcal{M} \equiv \bigwedge_{i=1}^{m} c_i$ un modelo booleano definido sobre un conjunto de features $\mathcal{F}$ y expresado como fórmula en CNF, donde toda c_i cumple que no es una tautología. Sea $e \in \mathtt{ent}(\mathcal{F})$ una entidad tal que $\mathcal{M}(e) = 1$. Se define un hipergrafo $H = (V, E)$ como:*

- $V := \mathcal{F}$
- $E := \{Var(c_i^e) : 1 \leq i \leq m\}$

Luego, $S \in SR(\mathcal{M}, e)$ si y solo si S es un transversal minimal de H. Además, $|H| = |\mathcal{F}| + \sum_{i=1}^{m} |c_i^e|$.

Demostración. Sea $S \subseteq \mathcal{F}$. Si para algún i vale que $S \cap Var(c_i^e) = \emptyset$, entonces como las cláusulas que definen a $\mathcal{M}$ no son tautologías, existe una entidad e_i tal que $c_i^e(e_i) = 0$. Considerar además la entidad e', definida de la siguiente manera:

$$e'(x) = \begin{cases} e(x) & \text{si } x \in \mathcal{F} \setminus Var(c_i^e) \\ e_i(x) & \text{en caso contrario} \end{cases}$$

Como $c_i^e(e') = 0$, entonces $\mathcal{M}(e') = 0$. Por lo tanto, $S \notin SR(\mathcal{M}, e)$.

Recíprocamente, suponer ahora que S es un hitting set de H. Sea e' una entidad que asigna los mismos valores que e a aquellas *features* en S. Luego, dada una cláusula c_i^e, existe una *feature* $x \in S \cap c_i^e$, y en consecuencia existe un literal en c_i^e tal que e' lo satisface. Por lo tanto, vale que $c_i^e(e') = 1$, y como esto es válido para cualquier i, debe valer que $\mathcal{M}(e') = 1$. □

También existe una relación entre los árboles de decisión binarios y los transversals (o hitting sets) minimales en hipergrafos, que será útil para las reducciones polinomiales que se presentarán en los resultados principales de este trabajo:

Lema 28 (Transversals minimales y árboles de decisión)**.** *Sea $H = (V, E)$ un hipergrafo. Es posible construir un árbol binario de decisión $T \in DT$ definido sobre el conjunto de features V y una entidad $e \in \mathbf{ent}(V)$ tales que el conjunto de transversals minimales de H coincida con $SR(T, e)$.*

Además, la construcción tiene complejidad temporal $O(|V|\,|E|)$, y T tiene tamaño $\Theta(|V|\,|E|)$.

Demostración. Se define la entidad e como $e(v) = 1$ para todo $v \in V$. Para cada hiperarista $B \in E$ se definen dos cláusulas:

$$c_B^+ \equiv \bigvee_{v \in B} v = 1$$
$$c_B^- \equiv \bigvee_{v \in V \setminus B} v = 0$$

Considerar el siguiente modelo:

$$\mathcal{M} \equiv \bigwedge_{B \in E} (c_B^+ \vee c_B^-)$$

Notar que toda cláusula $c_B^+ \vee c_B^-$ es satisfecha para todas las entidades menos para una. Por lo tanto, $\mathcal{M}$ rechaza una cantidad de $|E|$ entidades, con lo cual aplica el resultado del Lema 11: se puede construir el modelo $\neg\mathcal{M}$, y luego tomando el complemento de cada hoja se obtiene un árbol de decisión que representa a $\mathcal{M}$. En resumen, es posible representar a $\mathcal{M}$ como un árbol binario de decisión T, que satisface:

$$\mathtt{cnf}(T) = \bigwedge_{B \in E} (c_B^+ \vee c_B^-)$$

Luego:

$$\mathtt{cnf}(T)_e = \bigwedge_{B \in E} (c_B^+ \vee c_B^-)^e = \bigwedge_{B \in E} c_B^+$$

Entonces, por el Lema 27, las sufficient reasons para e dado $\mathcal{M}$ coinciden con los transversals minimales del hipergrafo $H' = (V, \{Var(c_B^e) : B \in E\})$; pero como $Var(c_B^e) = B$, entonces $H = H'$.

La complejidad enunciada es correcta ya que:

- La construcción del modelo $\mathcal{M}$ requiere construir las cláusulas c_B^+, c_B^- para todas las hiperaristas $B \in E$ (esto toma $O(|B|)$ tiempo y espacio para cada cláusula), y luego tomar la conjunción de todas ellas (una fórmula que tendrá tamaño $\Theta(|c_B^+| + |c_B^-)| = \Theta(|V|\,|E|)$).
- En el árbol de decisión a construir, se le asigna una hoja (un camino desde la raíz hasta esa hoja) a cada entidad rechazada por $\mathcal{M}$. Para garantizar que únicamente tal asignación termine en la hoja correspondiente, el camino debe consultar a las $|V|$ *features* (recordar que el árbol está definido sobre el conjunto de *features* V). Luego, por cada camino que se construye se cuentan $\Theta(|V|)$ vértices internos, siendo entonces $\Theta(|V|\,|E|)$ vértices en total en el árbol. A su vez, el tiempo total requerido para la construcción es proporcional a la cantidad de vértices creados.

□

Entonces, para cualquier hipergrafo H se puede construir un árbol de decisión binario T y una entidad e tal que los transversals minimales de H sean exactamente las sufficient reasons para e dado el modelo T.

Intuitivamente, se puede afirmar que es posible reducir el problema de encontrar las features relevantes (en árboles de decisión binarios) a *golpear* (*hit*) una serie de conjuntos; es decir, un problema de hitting set: los conjuntos a *golpear* serían las cláusulas de la representación del árbol T como fórmula CNF, y los elementos en el conjunto solución constituyen el conjunto de *features* relevantes. Esta intuición aparecerá cuando se discuta dicho problema en el siguiente capítulo.

Como es posible imaginar por los Lemas 27 y 28, resultará útil calcular transversals minimales para hipergrafos. Es por eso que se presentan los siguientes resultados, a partir de los cuales se deducen algoritmos para hallar estos transversals. Si bien existen algoritmos más eficientes para tareas similares, los algoritmos que se presentarán aquí sirven para demostrar resultados teóricos de tratabilidad bajo ciertas restricciones.

Lema 29. *Dado un hipergrafo $H = (V, E)$ y un subconjunto de vértices $W \subseteq V$, se puede decidir si existe un transversal minimal para H que contenga a W con complejidad temporal $O(|H| \, \Pi_{w \in W} deg(w))$.*

Demostración. Si $\mathcal{C}$ es un transversal minimal para H y $v \in \mathcal{C}$, entonces debe existir alguna hiperarista $B \in E$ tal que $\mathcal{C} \cap B = \{v\}$; si no fuera ese el caso, entonces $\mathcal{C} \setminus \{v\}$ sería también un transversal, y eso no es posible pues $\mathcal{C}$ es minimal por hipótesis. Luego, si $\mathcal{C}$ es un transversal tal que $W \subseteq \mathcal{C}$, entonces para cada $w \in W$ debe existir alguna hiperarista $B_w \in E$ donde $\mathcal{C} \cap B_w = \{w\}$.

El algoritmo consiste en una búsqueda exhaustiva en todas las posibilidades de elegir un $\{B_w\}_{w \in W} \subseteq E$, que son a lo sumo $\Pi_{w \in W} deg(w)$. Dada una de estas elecciones, es posible decidir si existe un transversal incluido en $V \setminus \left(\bigcup_{w \in W}(B_w \setminus \{w\})\right)$. Esto es lo mismo que verificar si $B \not\subseteq \bigcup_{w \in W}(B_w \setminus \{w\})$ para cada $B \in E$. Si vale esta verificación, entonces existe un transversal minimal de H contenido en $V \setminus \left(\bigcup_{w \in W}(B_w \setminus \{w\})\right)$, y éste debe contener a cada w dado que de no ser así, la hiperarista B_w no estaría siendo cubierta.

Dada una elección de $\{B_w\}_{w \in W}$, se puede decidir si $V \setminus \left(\bigcup_{w \in W}(B_w \setminus \{w\})\right)$ es un transversal de H en $O(|H|)$. Luego, la complejidad total del algoritmo es la enunciada. □

Lema 30. *Dado un hipergrafo $H = (V, E)$ y un vértice $v \in V$, es posible decidir si existen k transversal minimales distintos que contengan a v, con complejidad temporal $O(k \, |H| \, (deg(v) \, |V|^{k-1} + |V|))$.*

Demostración. Suponer que ya se han calculado $k - 1$ transversals minimales, a saber: $\mathcal{C}_1, \ldots, \mathcal{C}_{k-1}$. El siguiente transversal $\mathcal{C}_k$ se calcula de la siguiente forma: usando las ideas del Lema 29, se puede afirmar que si un conjunto de vértices $\mathcal{C}$ es un transversal minimal que contiene a v, entonces existe alguna hiperarista $B \in E$ tal que $\mathcal{C} \cap B = \{v\}$. Además, teniendo en cuenta que los k transversals deben ser distintos, debe ocurrir que $\mathcal{C}_k \neq \mathcal{C}_i$ para todo $1 \leq i \leq k - 1$. Entonces, tiene que existir para todo i algún vértice $s_i \in \mathcal{C}_i$ tal que $s \notin \mathcal{C}_k$.

El algoritmo consiste en una búsqueda exhaustiva con el mismo estilo que la presentada en el Lema 29: si existe algún otro transversal minimal $\mathcal{C}_k$ que contiene v, entonces debe existir alguna hiperarista $B \in E$, con $v \in B$ y deben existir también vértices $s_1, \ldots, s_{k-1}$

con $s_i \in \mathcal{C}_i$ tales que $\mathcal{C}_k \subseteq V \setminus ((B \setminus \{v\}) \cup \{s_1, \ldots, s_{k-1}\})$. Luego, para cada una de las $deg(v)$ elecciones posibles para B, y cada uno de los $|V|^{k-1}$ vértices, se verifica si $V \setminus ((B \setminus \{v\}) \cup \{s_1, \ldots, s_{k-1}\})$ es un transversal; en caso de que lo sea, se obtiene $\mathcal{C}_k$ removiendo vértices hasta obtener uno minimal. Esta última parte del algoritmo se puede completar en tiempo del orden de $O(|H|\,|V|)$. Por lo tanto, se puede encontrar $\mathcal{C}_k$ en $O(|H|\,deg(v)\,|V|^{k-1} + |H|\,|V|)$.

Como el algoritmo descrito aquí tenía como objetivo calcular el $k-$ésimo transversal minimal para H, luego, para calcularlos todos, se debe utilizar dicho algoritmo k veces, con lo cual vale la cota de complejidad del enunciado. □

3. RESULTADOS PRINCIPALES Y DISCUSIÓN

En esta sección, se estudiará la complejidad computacional de los problemas de encontrar *features* relevantes, necesarias y útiles para diferentes familias de modelos. El objetivo es identificar familias para las cuales estos problemas puedan resolverse de forma eficiente.

Al igual que los problemas definidos en 2.4, casi todos los que se formalizará aquí son de decisión (es decir, problemas que se responden con "sí" o "no", donde el objetivo es determinar si una cierta entrada satisface alguna proposición o condición). Además, los modelos que aparecerán en los diferentes problemas son genéricos; esto es, no se está forzando a que la representación del modelo sea una en específico (como puede ser un árbol de decisión, o una red neuronal). Es por eso que se eligió la notación $\texttt{EX}_{\mathcal{L}}$ para hacer referencia a un problema que no tiene un tipo de modelo fijo (en este ejemplo, el problema se llama `EX`, y se reemplazaría el subíndice "$\mathcal{L}$" por la familia de clasificadores correspondiente, e.g. $\texttt{EX}_{\texttt{DT}}$ sería el problema `EX` para árboles de decisión booleanos). Luego, se ahondará en detalle acerca de lo que ocurre si se fija el tipo de clasificador que se utiliza para representar un modelo, dado que la complejidad de estos problemas habitualmente depende de ello.

El primer problema a definir será el clásico problema de satisfacibilidad, aplicado a un modelo clasificador booleano. Como es habitual, este problema se utilizará como *punto de referencia* para las reducciones polinomiales, dado que su clase de complejidad es conocida para todos los modelos a considerar, y es habitualmente utilizado como *benchmark* para demostrar si un problema es difícil o no[1]. Se trata del problema donde se responde afirmativamente si existe una entidad (una asignación a las *features* booleanas) tal que el modelo responde afirmativamente para ésta:

PROBLEMA: $\text{SAT}_{\mathcal{L}}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$, definido sobre un conjunto de *features* $\mathcal{F}$. SALIDA: $\mathcal{M}$ es satisfacible (i.e. $\exists\, e \in \texttt{ent}(\mathcal{F}) \mid \mathcal{M}(e) = 1$)

Este problema está en NP para todos los clasificadores mencionados en la sección previa (y para todos los clasificadores razonables) debido a que todos ellos pueden evaluarse eficientemente (incluso en tiempo lineal) dada una asignación. Además, examinando todas las posibles asignaciones a las *features* del modelo $\mathcal{M}$, se puede encontrar una solución en tiempo exponencial con un algoritmo de fuerza bruta. Esta sería la solución más directa, pero si se fija el parámetro de la familia del modelo clasificador, es posible aprovechar sus propiedades y características para mejorar la cota de complejidad del problema. Por ejemplo, para algunos modelos relativamente simples como los árboles de decisión, o algunos con mayor *succinctness* como los FBDDs o los circuitos d-DNNF, el problema de decidir satisfacibilidad se puede resolver en tiempo polinomial [26].

[1] En este trabajo, se dirá que un problema es *difícil* cuando pertenezca a la clase de complejidad NP-HARD.

3.1. *Features relevantes* y *sufficient reasons*

Se definen ahora los primeros problemas de decisión que hacen a uno de los objetivos de esta tesis: relevancia de *features* y sufficient reasons. Empezando por el primero:

PROBLEMA: RELEVANT$_{\mathcal{L}}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \mathtt{ent}(\mathcal{F})$, y una *feature* $x \in \mathcal{F}$. SALIDA: x es relevante para e dado $\mathcal{M}$ (i.e. $Rel(x, \mathcal{M}, e)$)

Una instancia de RELEVANT$_{\mathcal{L}}$ es positiva si la *feature* x es *relevante* para la entidad e dado $\mathcal{M}$, lo que es lo mismo que decir que x pertenece a alguna sufficient reason de e dado $\mathcal{M}$. El punto de interés aquí es calcular las *features* relevantes para una entidad dado un modelo, y el problema de decisión asociado a este problema de búsqueda es el que se definió como RELEVANT$_{\mathcal{L}}$. Más adelante se discutirán otras propuestas de problemas que fueron surgiendo durante el transcurso de este trabajo, principalmente generalizaciones de este problema que apuntan a proponer nociones de explicaciones más generales.

Tanto en este como en los problemas que se estudiarán aquí, se trabaja con modelos booleanos en todos los sentidos (i.e. las *features* toman dos posibles valores, y las clases en las que puede ser clasificada una entidad también son dos), salvo cuando se indique lo contrario. Sin embargo, en general es posible extender las definiciones y resultados de complejidad a modelos con *features* con dominios más grandes y mayor cantidad de clases.

Antes de pasar al siguiente problema, se enuncian ciertas propiedades acerca de los modelos constantes:

Lema 31 (Caracterización de modelos booleanos constantes)**.** *Dado un modelo $\mathcal{M}$ definido sobre un conjunto de features $\mathcal{F}$, son equivalentes las siguientes afirmaciones:*

i) $\mathcal{M}$ *es constante.*

ii) Para toda entidad $e \in \mathtt{ent}(\mathcal{F})$, $SR(\mathcal{M}, e) = \{\emptyset\}$

iii) Existe una entidad $e \in \mathtt{ent}(\mathcal{F})$ *tal que* $SR(\mathcal{M}, e) = \{\emptyset\}$.

Demostración. Para probar que *i)* implica *ii)*, suponer que $\mathcal{M}$ es constante, i.e. para algún $b \in \{0, 1\}$, vale que $\forall\, e \in \mathtt{ent}(\mathcal{F}) : \mathcal{M}(e) = b$. Luego, dada una entidad e cualquiera, notar que el conjunto de *features* $X = \emptyset$ es sufficient reason para $\mathcal{M}(e)$, pues al fijar el valor de todas las *features* en X (es decir, ninguna) y dejando libre el valor del resto (todas), se tiene que $\mathcal{M}(e) = b$ y por más que se varíe el valor de cualquiera de las *features* que no están en X, el resultado seguirá siendo b. Por lo tanto, se concluye que $SR(\mathcal{M}, e) = \{X\} = \{\emptyset\}$.

Vale trivialmente que *ii)* implica *iii)*.

Para ver que *iii)* implica *i)*, suponer que $SR(\mathcal{M}, e) = \{\emptyset\}$ para alguna entidad $e \in \mathtt{ent}(\mathcal{F})$. Como $\emptyset$ es una sufficient reason para e dado $\mathcal{M}$, entonces si se cambia el valor de cualquier *feature* $x_0 \in \mathcal{F} \setminus \{\emptyset\} = \mathcal{F}$ (es decir, *cualquier feature*), entonces el resultado provisto por $\mathcal{M}$ no cambiará. Por lo tanto, el modelo es constante. □

A partir del resultado del Lema 31, se propone el siguiente problema de decisión:

PROBLEMA: SUFF-REASON$_{\mathcal{L}}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$, definido sobre un conjunto de *features* $\mathcal{F}$, y una entidad $e \in \texttt{ent}(\mathcal{F})$. SALIDA: Existe al menos una sufficient reason no vacía para e dado $\mathcal{M}$ (i.e. $SR(\mathcal{M}, e) \neq \{\emptyset\}$)

Este es el problema de la existencia de sufficient reasons, que, debido a la caracterización que se planteó en el Lema 31, es equivalente a decidir si un modelo es constante. Este problema puede parecer no relacionado con el cálculo de *features* relevantes, pero su utilidad será mostrada en los resultados siguientes.

3.1.1. Detección de relevancia

Las sufficient reasons para una entidad dado un modelo pueden diferir en todas sus *features* en peor caso, y puede haber una cantidad exponencial de ellas, respecto de la cantidad de *features*; entonces, quedarse con una sola sufficient reason para una entidad dada no alcanzaría para tener una explicación sólida. Y como se menciona en trabajos como [6], calcular la totalidad de las sufficient reasons para una entidad no suele ser viable en una cantidad razonable de tiempo. Además de esto, incluso si fuera viable, es probable que el tamaño del conjunto de sufficient reasons obtenido (o incluso de muchas sufficient reasons) sea tan grande, que no pueda ser interpretado por un individuo.

Luego, como encontrar una única sufficient reason no es suficiente, y encontrarlas todas no es viable o útil, es que se han propuesto acercamientos a la síntesis del conjunto de sufficient reasons. Es aquí donde cobran importancia las *features relevantes* (o irrelevantes, o necesarias), dado que sus cantidades no pueden ser más grandes que la cantidad de *features* total del modelo, y aún así pueden brindar cierto nivel de información útil acerca del mismo.

Se estudiará ahora la complejidad computacional del problema de decidir si una *feature* es relevante para una entidad dado un modelo (booleano). Para ello, se utilizarán reducciones a otros problemas conocidos. Concretamente, se demostrará que este problema (RELEVANT$_{\mathcal{L}}$) es tan difícil como SAT$_{\mathcal{L}}$. Para eso, considerar primero el siguiente resultado intermedio:

Proposición 32. *SAT$_{\mathcal{L}}$ $\leq_P$ SUFF-REASON$_{\mathcal{L}}$*

Demostración. Se define la función f que transforma instancias de SAT$_{\mathcal{L}}$ en instancias de SUFF-REASON$_{\mathcal{L}}$: dada una instancia de SAT$_{\mathcal{L}}$ $I = \mathcal{M}$, considerar la evaluación de $\mathcal{M}$ en la entidad $e = (1, \ldots, 1)$[2].

- si $\mathcal{M}(e) = 1$, entonces $\mathcal{M}$ es satisfacible. Por lo tanto, se define $f(\mathcal{M}) = (\mathcal{M}', e')$, donde $(\mathcal{M}', e')$ es una instancia donde vale trivialmente SUFF-REASON$_{\mathcal{L}}$[3].
- si $\mathcal{M}(e) = 0$, entonces se define $f(\mathcal{M}) = (\mathcal{M}, e)$. Por el Lema 31, si $\mathcal{M}$ no es constante, entonces tiene alguna sufficient reason no vacía, lo cual significa que la instancia de SUFF-REASON$_{\mathcal{L}}$ es positiva. Por otro lado, si $\mathcal{M}$ fuera constante, entonces sería insatisfacible, ya que evalúa a 0 para toda entidad. Luego, la instancia de SUFF-REASON$_{\mathcal{L}}$ es negativa.

[2] Recordar que la evaluación del modelo es posible en tiempo polinomial.
[3] Por ejemplo, alcanza con tomar $\mathcal{M}'(x_1, \ldots, x_n) = x_1$ y $e' = e$.

□

Para completar la reducción de $\text{SAT}_{\mathcal{L}}$ a $\text{RELEVANT}_{\mathcal{L}}$, el siguiente paso será probar que $\text{SUFF-REASON}_{\mathcal{L}} \leq \text{RELEVANT}_{\mathcal{L}}$. Notar la falta del subíndice P en el $\leq$: esto se debe a que esta reducción no resultó tan satisfactoria como la de la Proposición 32, ya que no fue posible hallar una reducción polinomial *usual* (una reducción de Karp); sin embargo, se consiguió dar con un a reducción más débil, que de todas formas sirve para el propósito de este trabajo: la llamada *reducción de tablas de verdad* (truth table reduction [64]). Este tipo de reducción consiste, en líneas generales, en describir la respuesta de un problema como una fórmula booleana o una tabla de verdad de una cantidad finita de consultas a otro problema.

Proposición 33. *$\text{SUFF-REASON}_{\mathcal{L}} \leq_{tt} \text{RELEVANT}_{\mathcal{L}}$*

Demostración. Dado un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$, y una entidad $e \in \texttt{ent}(\mathcal{F})$, se probará que la siguiente transformación de instancias es correcta:

$$\text{SUFF-REASON}_{\mathcal{L}}(\mathcal{M}, e) \equiv \bigvee_{x \in \mathcal{F}} \text{RELEVANT}_{\mathcal{L}}(\mathcal{M}, e, x)$$

⇒) Suponer que $\text{SUFF-REASON}_{\mathcal{L}}$ responde positivamente para $(\mathcal{M}, e)$. Entonces, existe al menos una sufficient reason no vacía. Luego, existe una *feature* x_0 tal que $x_0 \in X$ con $X \in SR(\mathcal{M}, e)$. Entonces, $\text{RELEVANT}_{\mathcal{L}}$ responde positivamente para al menos una de las instancias generadas: la instancia $(\mathcal{M}, e, x_0)$.

⇐) Suponer que $\text{RELEVANT}_{\mathcal{L}}$ responde positivamente para alguna instancia $(\mathcal{M}, e, x)$. Luego, existe $X \in SR(\mathcal{M}, e) \,|\, x \in X$, por definición de relevancia. Por lo tanto, como $X \neq \emptyset$, se tiene que $SR(\mathcal{M}, e) \neq \{\emptyset\}$, i.e. $\text{SUFF-REASON}_{\mathcal{L}}$ responde positivamente para $(\mathcal{M}, e)$.

Notar que la transformación cumple con la definición de las reducciones de tablas de verdad, dado que el problema $\text{SUFF-REASON}_{\mathcal{L}}$ se está reduciendo a una fórmula booleana que realiza $|\mathcal{F}|$ consultas a $\text{RELEVANT}_{\mathcal{L}}$. □

En conclusión, combinando los resultados obtenidos en las proposiciones 32 y 33, se puede afirmar que $\text{RELEVANT}_{\mathcal{L}}$ (y $\text{SUFF-REASON}_{\mathcal{L}}$) es tan difícil como $\text{SAT}_{\mathcal{L}}$; esto da lugar a que para modelos representados con estructuras donde decidir satisfacibilidad fuera difícil, también lo sería difícil decidir si una *feature* es relevante (o si existe alguna sufficient reason no vacía) para una entidad dada.

Como consecuencia, para cualquier familia de modelos $\mathcal{L}$ tal que $\text{SAT}_{\mathcal{L}}$ sea NP-COMPLETE, el problema $\text{RELEVANT}_{\mathcal{L}}$ probablemente no se pueda resolver en tiempo polinomial[4]. Este sería el caso cuando $\mathcal{L}$ sea la familia de circuitos booleanos generales.

Sin embargo, existen otras familias de modelos que, si bien no poseen el nivel de *succinctness* de los circuitos booleanos en general, sí son más potentes que un árbol de decisión booleano. Algunas de estas familias son aquellas basadas en los diagramas de decisión binarios, como OBDD y FBDD, o también los circuitos D-DNNF.

Durante la confección de este trabajo, se intentó arribar a resultados de complejidad para $\text{RELEVANT}_{\mathcal{L}}$ con dichas familias de modelos; pero fue en ese momento que se encontró el reciente artículo [39], donde se demuestra que el problema de decidir si una

[4] No se puede garantizar que $\text{RELEVANT}_{\mathcal{L}}$ sea también NP-COMPLETE ya que la reducción de la Proposición 33 no es de Karp.

feature es relevante resulta NP-HARD para FBDD. Como consecuencia, también resultará NP-HARD para cualquier familia de modelos con mayor *succinctness*, como es el caso de los circuitos D-DNNF según la jerarquía de [26]. Sin embargo, esto no diría nada acerca del problema $\text{RELEVANT}_{\texttt{OBDD}}$, dado que los OBDDs son una versión más restrictiva de los FBDDs. A pesar del tiempo y el esfuerzo dedicados a buscar un algoritmo polinomial para $\text{RELEVANT}_{\texttt{OBDD}}$, no fue posible obtener un resultado satisfactorio, con lo cual esta cota de complejidad queda abierta.

En resumen, el panorama para hallar algoritmos eficientes que decidan si una *feature* es o no relevante no resulta demasiado alentador, dado que es sabido que el problema de satisfacibilidad es intratable para la mayoría de los modelos más potentes, como por ejemplo las redes neuronales, o sin ir más lejos, los circuitos booleanos. Sin embargo, esto no derrota los objetivos de esta área de investigación (y de esta tesis), puesto que:

- Existen algunas estructuras de representación para las cuales decidir satisfacibilidad es tratable.
- Existen algoritmos (no polinomiales) para calcular *features* relevantes y sufficient reasons en diferentes tipos de modelos, los cuales se puede apuntar a optimizar.
- Existe la posibilidad de diseñar heurísticas para resolver estos problemas de manera más eficiente.
- El problema aquí presentado sólo recibe el modelo como entrada, pero en un escenario práctico se podría generar una estructura antes del uso del modelo[5] (e.g. una versión *compilada* del modelo, o algo similar a un índice en el mundo de las bases de datos), que permita responder las consultas de explicabilidad eficientemente, de manera similar a lo que ocurre con la compilación de circuitos [25, 53, 28, 41, 10].

3.1.2. Relevancia en árboles de decisión

Como se puede observar a partir de lo expuesto en esta sección del trabajo, detectar la relevancia de una *feature* es un problema difícil en general. Es por ello que se suele considerar este problema en el contexto de modelos relativamente simples, como árboles de decisión o familias restringidas de diagramas de decisión binarios [7, 39]. Por eso, ahora se estudiará el problema de calcular *features* relevantes en el caso particular de los árboles de decisión booleanos. Para ello, se comentará a grandes rasgos los avances realizados por otros autores en el estudio de la complejidad computacional del problema.

Los árboles de decisión tienen una característica distintiva en cuanto a la explicabilidad: existe una abductive explanation para clasificar cualquier instancia posible; ésta se conoce como *direct reason*: dado $T \in \text{DT}$ un árbol de decisión definido sobre un conjunto de features $\mathcal{F}$, y una entidad $e \in \texttt{ent}(\mathcal{F})$, la *direct reason* para e dado el modelo T es el término t, que corresponde al único camino desde la raíz hasta una hoja en T que es *compatible* con e; esto es, seguir el camino marcado por los valores de la entidad en cada vértice, empezando por la raíz y terminando en alguna hoja del árbol.

Las *direct reasons* pueden ser útiles para explicar una decisión tomada por un modelo de aprendizaje automático, pero tienen un problema en comparación con las sufficient reasons: una *direct reason* puede contener una cantidad arbitrariamente grande de *features redundantes* (es decir, no es minimal). Esta situación se observa frecuentemente en la

[5] Por ejemplo, durante la fase de entrenamiento del modelo.

práctica, incluso cuando el árbol de decisión es óptimo en tamaño [6, 43, 44]. Esto va en contra de uno de los principales objetivos de la explicabilidad: el proveer explicaciones que resulten claras y entendibles para aquel individuo a quien le corresponda entenderlas. Si se trata de un modelo con una gran cantidad de *features* (posiblemente en el orden de las miles, o incluso millones de ellas, lo cual no sería raro hoy en día), esta redundancia en las explicaciones haría que contar con ellas no resulte demasiado provechoso, dado que sería extremadamente complicado interpretarlas para un ser humano. Es por esto que, incluso existiendo una explicación intrínseca para cada entidad en un árbol de decisión booleano, sigue siendo conveniente investigar acerca del cálculo de explicaciones más pequeñas, como pueden ser las sufficient reasons. Por otro lado, la principal ventaja de las *direct reasons* es que son únicas, lo cual tiene sentido, dado que hay un único camino que es determinado completamente por una entidad.

Otra propiedad útil de los árboles de decisión es que están *cerrados por la negación* (además de por la conjunción, la disyunción exclusiva y el condicionamiento). En consecuencia, el considerar a un árbol T o a su negación no tiene un impacto computacional dado que $\neg T$ se puede calcular eficientemente en el tamaño de T. Es por ello que cuando se trate de árboles de decisión, es posible centrar la atención únicamente en explicaciones para instancias positivas del modelo, sin perder generalidad (esto también sucede con los OBDDs).

A diferencia de las *direct reasons*, que son únicas, la cantidad de sufficient reasons puede ser exponencial [6]. Existen diversos algoritmos para calcular todas las sufficient reasons para una entidad dado un árbol (se puede ver [6] un racconto sobre los mismos).

En [6] se muestra un algoritmo para encontrar *features* relevantes para una entidad $e \in \mathtt{ent}(\mathcal{F})$ dado un modelo representado como un árbol de decisión T definido sobre $\mathcal{F}$. Los resultados y reflexiones que se presentan en esta tesis pretenden sumar detalles y aclarar cuestiones que fueron discutidas en menor profundidad por los autores. En dicho trabajo, se aprovecha la posibilidad de representar un árbol de decisión como una fórmula en CNF, de una manera que se puede formalizar como se vio en la Definición 10. Una vez obtenida la representación en CNF del árbol de decisión T, se realiza una transformación que consiste en remover aquellos literales que sean *incompatibles* con la entidad e: si $e(x_i) = 1$ se borran los literales de la forma $\overline{x_i}$, y si $e(x_i) = 0$ se borran los de la forma x_i. La eliminación de estos literales incompatibles produce como resultado una fórmula en CNF que además resulta monótona (ver Definición 1).

Luego, a partir de la fórmula en CNF, existe un procedimiento para construir un hipergrafo cuyos transversals minimales coincidan con las sufficient reasons de e dado T. Los autores de [6] no ahondan en las formalidades del procedimiento, por lo que en el Lema 28 se muestra un resultado para explicarlo, del cual se puede deducir el algoritmo presentado en dicho trabajo.

Las nociones de dualidad entre las sufficient reasons y los transversals minimales en hipergrafos (o hitting sets minimales), cuyas formalizaciones fueron mostradas en la sección 2.4 de esta tesis, son aprovechadas en [6] para mostrar que $\text{RELEVANT}_{\mathtt{DT}}$ resulta tratable, y el algoritmo para resolverlo surge de las ideas presentes en el Lema 27[6]:

Teorema 34. *El problema* $\mathit{RELEVANT}_{\mathtt{DT}}$ *se puede resolver con complejidad temporal* $O(|\mathcal{F}|\,|T|^2)$.

[6] Sin embargo, en dicho artículo no se ahonda en tanto detalle como aquí, y a causa de ello se presenta como un algoritmo con complejidad $O((|\mathcal{F}| + |T|)\,|T|)$, la cual es incorrecta según los resultados que se muestran en esta tesis.

Demostración. Usando el Lema 27, se puede construir un hipergrafo H a partir de $\texttt{cnf}(T)^e$, tal que la *feature* x sea relevante para $T(e)$ si y solo si x pertenece a un transversal minimal de H. Esto se puede decidir en tiempo del orden de $O(|H|\, deg(v)) = O(|\mathcal{F}|\,|T|^2)$ usando el algoritmo del Lema 29. □

Por último, en [17] se extienden algoritmos para árboles de decisión a variaciones con *features* categóricas no necesariamente booleanas, e incluso *features* numéricas. Además, se consideran generalizaciones para las cuales se prueba pertenencia a la clase NP-HARD en el caso general, pero tratabilidad bajo ciertas hipótesis.

3.1.3. Sufficient reasons con estructura

La relevancia de *features* es un concepto fundamental en el área de la explicabilidad basada en lógica; sin embargo, muchas *features* pueden ser relevantes para una misma predicción, y como consecuencia, la relevancia puede no resultar un indicador lo suficientemente fuerte como para ordenar las *features* según ella (es en parte por eso que se planteó este trabajo desde un punto de vista basado en *feature selection* y no en *feature ranking*).

En consecuencia, se propone una generalización para el problema de la detección de *features* relevantes. La idea será encontrar sufficient reasons con una forma en especial; esto es, conjuntos de *features* que cumplan con la definición de sufficient reason, pero que contengan a una o más *features* específicas.

La utilidad de este problema es visible al considerar un escenario en el que un individuo quiera comprender el comportamiento de un modelo de inteligencia artificial, y que busque que las explicaciones que el modelo le pueda brindar para ello cumplan con ciertos requerimientos. Por ejemplo, en el caso de uso de un modelo de diagnóstico asistido por inteligencia artificial, un profesional de la salud podría querer conocer qué tanta influencia tiene un cierto síntoma (o conjunto de síntomas) en el diagnóstico de alguna enfermedad. Para eso, puede solicitar al sistema que le otorgue explicaciones (en forma de sufficient reasons) que contengan la(s) *feature*(s) que representa a ese síntoma.

Por otro lado, un usuario podría consultar por sufficient reasons con más de un elemento, con lo cual, de cierta manera, se está preguntando si ciertos conjuntos de *features* actúan eficazmente, de forma minimal y en conjunto, para realizar la predicción del modelo. Esta es una capacidad que escapa a la noción de *features* relevantes, pero que las sufficient reasons sí pueden captar.

Explicada la motivación de este problema, se procede a definirlo concretamente:

> PROBLEMA: RESTRICTED-SR$_{\mathcal{L}}$
> ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \texttt{ent}(\mathcal{F})$, y un conjunto de *features* $X \subseteq \mathcal{F}$.
> SALIDA: Existe una sufficient reason S para e dado $\mathcal{M}$ tal que X está contenido en S. $(\exists\, S \in SR(\mathcal{M}, e) \mid X \subseteq S)$

Observación 35. *El problema RESTRICTED-SR$_{\mathcal{L}}$ es equivalente a RELEVANT$_{\mathcal{L}}$ en el caso en que $|X| = 1$. Por lo tanto, RESTRICTED-SR$_{\mathcal{L}}$ es una generalización de RELEVANT$_{\mathcal{L}}$, con lo cual ya se puede afirmar en este punto que este problema será difícil para familias de modelos desde FBDD en adelante (en un sentido de succinctness de modelos[7]), por lo explicado en 3.1.1.*

[7] Esto incluye, por ejemplo, a los circuitos booleanos generales.

Se estudiará entonces el caso particular de RESTRICTED-SR$_{\mathcal{L}}$ para árboles de decisión booleanos; es decir, el problema RESTRICTED-SR$_{\mathtt{DT}}$. Los resultados de complejidad para este problema tendrán relación con los transversals minimales de hipergrafos, al igual que ocurrió en el caso de RELEVANT$_{\mathtt{DT}}$ en 3.1.2. Estos conceptos fueron formalmente presentados en los Lemas 27 y 28, donde se destacó que el cálculo de transversals minimales (en este caso, con la restricción adicional de que éstos deben contener ciertos vértices correspondientes al conjunto X) forma parte esencial de la búsqueda de sufficient reasons.

Notar que el algoritmo que surge a partir del resultado presentado en el Lema 29 no resulta polinomial en el tamaño de la entrada; de hecho, esto no es mejorable en el caso general, dado que el problema es NP-COMPLETE. En particular, se mostrará esto para RESTRICTED-VERTEX-COVER, el equivalente en grafos (y por lo tanto, menos difícil) del problema del hitting set (o transversal) minimal en hipergrafos.

Proposición 36. *El problema RESTRICTED-VERTEX-COVER es NP.*

Demostración. Considerar una instancia afirmativa del problema y un certificado de la misma. El objetivo es verificar este último en tiempo polinomial respecto del tamaño de la entrada. Un certificado para RESTRICTED-VERTEX-COVER tendrá la forma de un conjunto de vértices $\mathcal{C}$. Para verificar que es un certificado afirmativo, hace falta comprobar que:

- $\mathcal{C}$ es un vertex cover: para esto, se puede recorrer el conjunto de aristas del grafo $G = (V, E)$, y verificar que alguno de los dos extremos de cada arista pertenezca a $\mathcal{C}$. Esto se puede hacer en $O(|V|\,|E|)$, dado que el tamaño máximo del $\mathcal{C}$ es $|V|$ (en peor caso, el vertex cover tendría tantos vértices como el grafo), y la verificación se hace a lo sumo dos veces por cada arista.

- $\mathcal{C}$ es minimal como vertex cover: basta con remover iterativamente un vértice de $\mathcal{C}$, y comprobar que el resultado sea un vertex cover con el procedimiento descrito en el punto anterior. Esto tiene una complejidad de $O(|V|\,(|V|\,|E|))$, i.e. $O(|V|^2\,|E|)$.

- $W \subseteq \mathcal{C}$: simplemente verificar para cada $v \in W$, si $v \in \mathcal{C}$. Esto tiene una complejidad de $O(|V|^2)$.

Es inmediato concluir que la complejidad de verificar el certificado para el problema RESTRICTED-VERTEX-COVER es polinomial respecto al tamaño de entrada. □

En el siguiente resultado se demostrará que RESTRICTED-VERTEX-COVER es un problema NP-HARD. Para ello se recurrirá a un problema estándar de coloreo de grafos: 3-COLOR, que responde positivamente si un grafo admite un 3-coloreo. Es conocido el hecho de que 3-COLOR es un problema NP-COMPLETE para grafos generales.

Proposición 37. *El problema RESTRICTED-VERTEX-COVER es NP-HARD.*

Demostración. Se mostrará que existe una reducción polinomial desde 3-COLOR al problema RESTRICTED-VERTEX-COVER, pasando por dos problemas intermedios: primero 3,2-CNF-MONOTONE-SAT y luego HITTING-SET-FP.

1. 3-COLOR $\leq_P$ 3,2-CNF-MONOTONE-SAT: Sea $G = (V, E)$ un grafo y $C = \{1, 2, 3\}$ el conjunto de colores. Por cada vértice $v \in V$ y color $c \in C$, se definen las variables $x_{v,c}$. La intuición es que si $x_{v,c}$ es verdadera para una valuación dada,

entonces que el vértice v tiene asignado el color c; caso contrario, el vértice no tiene asignado ese color. Es decir, dada una valuación de las variables se puede definir un coloreo f de los vértices como $f(v) = c \iff x_{v,f(v)} = 1$. Para que esta regla defina un coloreo válido (es decir, que f esté bien definida y vértices adyacentes tengan colores distintos) se imponen las siguientes restricciones:

(α) todo vértice debe tener algún color.

$$\alpha \equiv \bigwedge_{v \in V} (x_{v,1} \vee x_{v,2} \vee x_{v,3})$$

(β) dos vértices adyacentes no pueden tener el mismo color.

$$\beta \equiv \bigwedge_{(v,w) \in E, c \in C} (\neg x_{v,c} \vee \neg x_{w,c})$$

(γ) un vértice no puede tener dos colores simultáneamente.

$$\gamma \equiv \bigwedge_{c,d \in C, c \neq d} (\neg x_{v,c} \vee \neg x_{v,d})$$

Observar que α es de la forma de ϕ^+, y β y γ son de la forma de ϕ^- en el contexto de 3,2-CNF-MONOTONE-SAT. Por lo tanto, se afirma que $\phi^+ \equiv \alpha$ y $\phi^- \equiv \beta \wedge \gamma$, siendo $\phi \equiv \phi^+ \wedge \phi^-$.

Ahora se probará que esta transformación es correcta:

$\Rightarrow$) Suponer que G es 3-coloreable. Entonces, existe una función $f\colon V \to C$ tal que $\forall (v,w) \in E : f(v) \neq f(w)$. Dada f, se define una valuación $\sigma : \{x_{v,c}\}_{v \in V, c \in C} \to \{0,1\}$ para las variables de la fórmula como $\sigma(x_{v,c}) = 1 \iff c = f(v)$. Se verá que esta valuación satisface a $\alpha \wedge \beta \wedge \gamma$:

- (α) Para cada $v \in V$ exactamente uno de los tres literales $x_{v,1}, x_{v,2}, x_{v,3}$ vale 1 bajo σ, de modo que α se cumple.
- (β) Sean $v, w \in V$ tales que $(v,w) \in E$. Suponer que $\neg\sigma(x_{v,c}) \vee \neg\sigma(x_{w,c}) = 0$. Es decir $\sigma(x_{v,c}) = \sigma(x_{w,c}) = 1$. Entonces, por definición de σ, se tiene que $f(v) = c$. Equivalentemente, $f(w) = c$. Por lo tanto, vale que $f(v) = f(w)$, pero eso es absurdo, ya que $(v,w) \in E$ y f es un coloreo de G. La contradicción provino de haber supuesto que $\neg\sigma(x_{v,c}) \vee \neg\sigma(x_{w,c}) = 0$. Luego, debe valer β.
- (γ) Para un vértice fijo v no pueden valer dos colores a la vez porque la definición de la valuación σ asigna 1 a un único color y 0 a los otros dos. Por tanto, $\neg\sigma(x_{v,c}) \vee \neg\sigma(x_{v,d}) = 1$ para todo par $c \neq d$, i.e. vale γ.

$\Leftarrow$) Suponer ahora que ϕ es satisfacible. Sea σ una valuación que la hace verdadera. Luego:

- *Existencia de color.* Por α, para cada $w \in V$ al menos uno de los literales $x_{w,1}, x_{w,2}, x_{w,3}$ vale 1 bajo σ.
- *Unicidad de color.* Por γ, para cada vértice w no pueden valer simultáneamente dos de esos literales. Luego existe exactamente un $c \in C$ con $\sigma(x_{w,c}) = 1$.

Estos dos hechos confirman que existe una función $f\colon V \to C$ tal que:

$$f(w) = c \iff \sigma(x_{w,c}) = 1.$$

Por último, como vale también β, entonces para toda arista $(v, w) \in E$, se cumple que $\sigma(x_{v,c}) = 0 \ \vee \ \sigma(x_{w,c}) = 0$, para cada color $c \in C$. Entonces, tomando la función mencionada, se tiene que $f(v) = c = f(w)$ si y sólo si $\sigma(x_{v,c}) = \sigma(x_{w,c}) = 1$. Pero, por β, eso no puede pasar.
En conclusión, por la definición de coloreo, se tiene que f es un 3-coloreo válido para G.

2. 3,2-CNF-MONOTONE-SAT $\leq_P$ HITTING-SET-FP: Se propone una reducción guiada por la siguiente intuición: las *bolsas* B_i de la instancia de HITTING-SET-FP se corresponden con la parte no negada de ϕ, y los pares prohibidos p_j se corresponden con la parte negada. Es decir, dada una fórmula

$$\psi \equiv \bigwedge_{i=1}^{m} \left(x_1^i \vee x_2^i \vee x_3^i\right) \ \wedge \ \bigwedge_{j=1}^{k} \left(\neg y_1^j \vee \neg y_2^j\right) \tag{3.1}$$

se define la instancia de HITTING-SET-FP como $I = (X, \{B_i\}_{1\leq i\leq m}, \{p_j\}_{1\leq j\leq k}) = (\{x_\ell^i : 1 \leq \ell \leq 3,\ 1 \leq i \leq m\}, \{\{x_1^i, x_2^i, x_3^i\} : 1 \leq i \leq m\}, \{(y_1^j, y_2^j) : 1 \leq j \leq k\})$.

A partir de ahora, se llamará α a la primera fórmula CNF en (3.1) de ψ, y β a la segunda.

Se probará ahora que la reducción es correcta:

⇒) Suponer que se tiene una instancia positiva de 3,2-CNF-MONOTONE-SAT. Es decir, que la fórmula φ es satisfacible. Entonces, existe una valuación σ que satisface a φ. Como $\varphi = \alpha \wedge \beta$, entonces debe ocurrir que $\sigma(\alpha) = 1 \wedge \sigma(\beta) = 1$[8]. Esto es lo mismo que $(\sigma(x_1^i) = 1 \vee \sigma(x_2^i) = 1 \vee \sigma(x_3^i) = 1) \ \wedge \ (\sigma(y_1^j) = 0 \vee \sigma(y_2^j) = 0),\ \forall 1 \leq i \leq m,\ 1 \leq j \leq k$. Se define entonces un hitting set S para I como $S = \{x_s^i : \sigma(x_s^i) = 1\}$.
Cada *bolsa* B_i cumple que $S \cap B_i \neq \emptyset$ porque existe un x_s^i tal que $\sigma(x_s^i) = 1$. Por otro lado, como no hay ningún par y_1^j, y_2^j tal que $\sigma(y_1^j) = 1 \wedge \sigma(y_2^j) = 1$, también vale que $S \cap \{y_1^j, y_2^j\} \neq \{y_1^j, y_2^j\}$. Luego, la instancia I es positiva para HITTING-SET-FP.

⇐) Suponer ahora que se tiene una instancia positiva de HITTING-SET-FP, siendo S el hitting set de la solución. Entonces, para cada *bolsa* $B_i = \{x_1^i, x_2^i, x_3^i\}$, hay al menos uno de los x_s^i que pertenece al hitting set S. Por otro lado, para cada par $p_j = \{y_1^j, y_2^j\}$, no puede ocurrir que ambos elementos del par prohibido pertenezcan a S.
Entonces, tomando como valuación σ aquella que se comporte de la siguiente manera:

$$\sigma(x) = 1 \iff x \in S$$

[8] La valuación se define sobre las variables y no sobre las fórmulas, pero se hace este abuso de notación por conveniencia.

se tiene una asignación que, por construcción de la fórmula, satisface a φ. Por lo tanto, como la fórmula es satisfacible, la instancia de 3,2-CNF-MONOTONE-SAT es positiva.

3. HITTING-SET-FP $\leq_P$ RESTRICTED-VERTEX-COVER:

   Considerar la siguiente transformación de instancias de HITTING-SET-FP a instancias de RESTRICTED-VERTEX-COVER. Dada una instancia del problema HITTING-SET-FP definida como $I = (X, \{B_i\}_{1 \leq i \leq m}, \{(y_1^j, y_2^j)\}_{1 \leq j \leq k})$, se define el grafo $G = (V, E)$ de la instancia de RESTRICTED-VERTEX-COVER como:

   - $V = V_1 \cup V_2$, donde:
     - $V_1 := \bigcup_{i=1}^{m} B_i$ (un vértice por cada elemento de las *bolsas*)
     - $V_2 := \{u_i : 1 \leq i \leq m\}$ (un vértice por cada *bolsa*)
   - $E = E_1 \cup E_2$, donde:
     - $E_1 := \bigcup_{j=1}^{k} (y_1^j, y_2^j)$ (una arista entre cada uno de los pares p_j)
     - $E_2 := \bigcup_{i=1}^{m} \{(u_i, x) : x \in B_i\}$ (una arista entre cada vértice de V_2 y todos los vértices correspondientes a su *bolsa*)

   Se toma $W = V_2$ en la instancia propuesta de RESTRICTED-VERTEX-COVER. Ahora se probará que esta transformación es correcta. Sean $\mathcal{B} = \bigcup_{i=1}^{m} B_i$, y $\mathcal{P} = \bigcup_{j=1}^{k} p_j$ donde $p_j = (y_1^j, y_2^j)$.

   $\Rightarrow$) Suponer que se tiene una instancia positiva de HITTING-SET-FP. Sea $\mathcal{C} = V \setminus S$, donde S es el hitting set correspondiente a esta instancia. Vale que $W \subseteq \mathcal{C}$. Se demostrará que $\mathcal{C}$ es un vertex cover:

   - Sea $(v, w) \in E_1$: suponer que $v, w \notin \mathcal{C} = V \setminus S$. Luego, $v, w \in S$. Esto es absurdo, ya que como $(v, w) \in E_1$, entonces se trata de un par $(v, w) \in \mathcal{P}$, por lo que es imposible que estos elementos estén en S, el hitting set que no incluye pares prohibidos. En conclusión, $v \in \mathcal{C} \vee w \in \mathcal{C}$, i.e. $\mathcal{C}$ cubre todas las aristas de E_1.
   - Sea $(v, w) \in E_2$. Es decir, (v, w) es de la forma (u_i, x). Por definición, $u_i \in V_2$, con lo cual $u_i \notin S$, lo que es lo mismo que decir que $u_i \in \mathcal{C}$ para todo i. En conclusión, $\mathcal{C}$ cubre todas las aristas de E_2.

   Como $\mathcal{C}$ es un vertex cover, vale que existe un vertex cover minimal $\mathcal{C}' \subseteq \mathcal{C}$. Se verá entonces que todos los vértices $u_i \in V_2$ necesariamente deben formar parte de $\mathcal{C}'$: sea $u_i \in V_2$; los vecinos de u_i son aquellos vértices $x \in B_i$, por definición de V_2. Alguno de todos estos vértices pertenece a S, dado que es un hitting set. Sin pérdida de generalidad, se determina que $\hat{x} \in S \cap B_i$. Como $\mathcal{C} = V \setminus S$, entonces $\hat{x} \notin \mathcal{C}$. Sin embargo, existe una arista $(u_i, \hat{x}) \in E$. Con lo cual, necesariamente debe ocurrir que $u_i \in \mathcal{C}'$.

   Por lo tanto, la instancia resultante de transformar la instancia positiva de HITTING-SET-FP es una instancia positiva de RESTRICTED-VERTEX-COVER.

   $\Leftarrow$) Se supone ahora que hay una instancia positiva de RESTRICTED-VERTEX-COVER. Sea $\mathcal{C}$ un vertex cover minimal de G tal que $W \subseteq \mathcal{C}$. Se define entonces el conjunto $S = \mathcal{B} \setminus \mathcal{C}$. Se probará ahora que este es un hitting set sin pares

prohibidos: hay que demostrar primero que S contiene un elemento de cada uno de los B_i. Es decir, que $S \cap B_i \neq \emptyset \forall 1 \leq i \leq m$. Para esto, suponer lo contrario: que para un $B_\ell \in \mathcal{B}$ arbitrario, ocurre que $S \cap B_\ell = \emptyset$. Entonces, debe ocurrir que para todo elemento x de B_ℓ, valga que $x \in \mathcal{C}$. Además, por definición de $\mathcal{C}$, u_ℓ pertenece a $\mathcal{C}$. Luego, como tanto u_ℓ y todos sus vecinos están en el cubrimiento $\mathcal{C}$, entonces $\mathcal{C}$ no es minimal. Esto es un absurdo, proveniente de haber supuesto que $S \cap B_\ell = \emptyset$. Como la elección de B_ℓ fue arbitraria, se concluye que $S \cap B_i \neq \emptyset \forall 1 \leq i \leq m$.

Resta entonces demostrar que no hay ningún par prohibido $p_j \in \mathcal{P}$ cuyos dos elementos pertenezcan al conjunto S. Suponer lo contrario: sea $(y_1, y_2) \in \mathcal{P}$ tal que $y_1, y_2 \in S = \mathcal{B} \setminus \mathcal{C}$. Luego, $y_1, y_2 \notin \mathcal{C}$. Ahora, por definición de E_1, se tiene que $(y_1, y_2) \in E$, y por lo tanto la arista (y_1, y_2) no está cubierta por $\mathcal{C}$. Esto es un absurdo, dado que se trata de un cubrimiento de aristas por vértices. La contradicción proviene de suponer que existe un $(y_1, y_2) \in \mathcal{P}$ tal que $y_1, y_2 \in S = \mathcal{B} \setminus \mathcal{C}$. En conclusión, el conjunto S cumple con las restricciones impuestas por el conjunto de pares prohibidos de $\mathcal{P}$.

Por lo tanto, la instancia resultante de transformar la instancia positiva de RESTRICTED-VERTEX-COVER es una instancia positiva de HITTING-SET-FP.

□

Las proposiciones 36 y 37 dan lugar directamente a la siguiente consecuencia:

Corolario 38. *El problema RESTRICTED-VERTEX-COVER es NP-COMPLETE.*

Por un lado, esto es un resultado auxiliar para el objetivo de dar con una cota para la complejidad temporal de RESTRICTED-SR$_{\texttt{DT}}$. Sin embargo, también demuestra la intratabilidad de un problema de grafos (hallar un vertex cover minimal que contenga a ciertos vértices en particular), sobre lo cual no se encontró referencia en la bibliografía consultada. Más adelante, ya habiendo terminado esta tesis, y gracias al aporte hecho por Bernardo Subercaseaux como comentario a [17], se conoció [19], un trabajo reciente que, entre otras cosas, estudia la complejidad de la extensión de un subconjunto de vértices a un vertex cover minimal.

Por otro lado, se presentaron los Lemas 29 y 30 para mostrar que estos problemas, aunque intratables en el caso general (es decir, cuando $|W|$ o k no están fijos), podrían ser resueltos eficientemente en algunos casos más particulares. A su vez, estos algoritmos no serán óptimos para la tarea de enumerar hitting sets (o transversals) minimales: para ello, sería conveniente recurrir a alternativas más eficientes, como por ejemplo la presentada en [33].

Volviendo a RESTRICTED-SR$_{\texttt{DT}}$, se concluye lo siguiente sobre su complejidad computacional:

Teorema 39. *El problema RESTRICTED-SR$_{\texttt{DT}}$ es NP-COMPLETE, y se puede resolver con una complejidad temporal de $O(|\mathcal{F}|\,|T|^{|X|+1})$.*

Demostración. Para ver que el problema es NP, basta con mostrar que se puede verificar un certificado para una instancia afirmativa en tiempo polinomial respecto al tamaño de entrada. Este certificado será un conjunto de *features* S, para el cual hay que verificar que:

- S es una sufficient reason para $T(e)$: eso se puede hacer en tiempo polinomial para árboles de decisión, como se muestra en [39].
- $X \subseteq S$: esto también se puede hacer en tiempo polinomial, dado que consiste en verificar que cada elemento en X pertenezca a S.

Así queda probado que RESTRICTED-SR$_{\mathtt{DT}} \in$ NP. Faltaría entonces mostrar que también es NP-HARD. Esto es relativamente inmediato a partir de los resultados auxiliares: usando el Lema 28, se puede reducir el problema de calcular un hitting set minimal que contenga a ciertos vértices (que es NP-HARD por el Lema 30) al problema RESTRICTED-SR$_{\mathtt{DT}}$.

Por último, aplicar los Lemas 27 y 30 para obtener un algoritmo con la cota de complejidad enunciada. □

A modo de cierre para esta generalización, y a raíz de comentarios hechos acerca del resultado equivalente al Teorema 39 en [17][9], se puede observar que este teorema tiene relación con el Teorema 1 de [3] y con algunos conceptos de [5].

3.1.4. Features k-relevantes

Se presentará una nueva generalización para el problema de decidir si una *feature* es relevante: se trata del problema K-RELEVANT$_{\mathcal{L}}$. En este caso, no resulta un problema de decisión, sino uno de conteo:

PROBLEMA: K-RELEVANT$_{\mathcal{L}}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \mathtt{ent}(\mathcal{F})$, y una *feature* $x \in \mathcal{F}$. SALIDA: Cantidad de sufficient reasons para e dado $\mathcal{M}$ que contienen a x.

Este problema es interesante porque puede dar una noción de importancia más completa a las *features* de un modelo, en el sentido de que a cuantas más sufficient reasons pertenezca una *feature*, mayor será su importancia. Esto no será necesariamente siempre así, pero sí resulta de interés como propuesta.

Notar que con la introducción de este concepto, es natural pensar en un ordenamiento para las *features* de un modelo dada una entidad según la cantidad de sufficient reasons a las que pertenezcan. Es por eso que se puede decir que un método de explicabilidad basado en K-RELEVANT$_{\mathcal{L}}$ podría categorizarse como *feature ranking*, de manera similar a aquellos métodos basados en el *Shapley value*. Un posible trabajo de experimentación a futuro es comprobar si este tipo de *feature ranking* es o no efectivo, dado que la eficacia de esta propuesta no ha sido verificada.

De forma similar a la utilizada en el caso de RESTRICTED-SR$_{\mathcal{L}}$, se encarará el estudio de la complejidad de K-RELEVANT$_{\mathcal{L}}$ en el caso particular en que $\mathcal{L} =$ DT:

Teorema 40. *El problema K-RELEVANT$_{\mathtt{DT}}$ es #P-COMPLETE.*

Demostración. La propiedad de *hardness* de K-RELEVANT$_{\mathtt{DT}}$ surge a partir de una reducción del problema de contar la cantidad de vertex covers minimales de un grafo, que resulta #P-COMPLETE según [59][Teorema 8].

[9] En concreto, se agradece nuevamente el aporte de Bernardo Subercaseaux como retroalimentación a [17].

Sea $G = (V, E)$ un grafo, y sean $v, w \notin V$ dos vértices cualesquiera que no estén en G. Considerar el siguiente grafo $G' = (V \cup \{v, w\}, E \cup \{vw\})$.

Si $\mathcal{C} \subseteq V$ es un vertex cover minimal de G, entonces al agregar dos vértices aislados de V pero unidos entre ellos por la arista (v, w), es necesario agregar v (o bien w) a $\mathcal{C}$ para obtener un vertex cover minimal de G'. Luego, en particular es cierto que la cantidad de vertex covers minimales de G es igual que la cantidad de vertex covers minimales de G' que contienen a v.

Entonces, usando el Lema 28 se tiene que es posible construir un árbol de decisión booleano $T_{G'} \in \mathrm{DT}$ a partir de G', y una entidad $e \in \mathtt{ent}(\mathcal{F})$ tales que la cantidad de vertex covers minimales de G' que contienen a v sea igual que la cantidad de sufficient reasons para $T_{G'}(e)$ que contienen a v. □

A partir de K-RELEVANT$_{\mathcal{L}}$ se puede definir su problema de decisión asociado: el de decidir si existen al menos k sufficient reasons que contengan a cierta *feature*.

PROBLEMA: K-RELEVANT-DEC$_{\mathcal{L}}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \mathtt{ent}(\mathcal{F})$, una *feature* $x \in \mathcal{F}$ y un número natural k. SALIDA: Existen al menos k sufficient reasons para e dado $\mathcal{M}$ que contienen a x.

Un algoritmo eficiente para resolver este problema sería de utilidad para filtrar las *features* de un modelo, eligiendo únicamente aquellas que tengan cierto grado de *relevancia*; esto se puede representar como aquellas *features* para las cuales K-RELEVANT-DEC$_{\mathcal{L}}$ responda positivamente para cierto valor de k.

Con respecto a los algoritmos para resolver el caso particular de K-RELEVANT-DEC$_{\mathtt{DT}}$, se tiene el siguiente resultado que se desprende de otros resultados previos:

Proposición 41. *Existe un algoritmo que resuelve K-RELEVANT-DEC$_{\mathtt{DT}}$ en tiempo $O(k\,|\mathcal{F}|^{k+2}\,|\mathcal{M}|)$, donde $\mathcal{M} \in DT$.*

Demostración. Es consecuencia de aplicar los Lemas 27 y 30. □

Si k es un valor grande, este algoritmo no será eficiente; sin embargo, para valores de k chicos se puede resolver el problema con un tiempo de ejecución razonable.

Relacionado con esto último, y a pesar de que tanto estos problemas como el anteriormente estudiado RESTRICTED-SR$_{\mathtt{DT}}$ sean intratables en general, los resultados algorítmicos de los teoremas 39 y 40 sugieren que existen algoritmos eficientes generalizando la noción de relevancia para árboles de decisión. En el caso de los modelos con mayor *succinctness* que los FBDDs, estos problemas serán intratables dado que así también lo es RELEVANT$_{\mathtt{FBDD}}$. Por último, al igual que ocurre con RELEVANT$_{\mathtt{OBDD}}$, las versiones de los problemas generalizados para OBDDs siguen siendo un interrogante en cuanto a su complejidad.

3.2. *Features necesarias*

Ahora se estudiará la complejidad de decidir si una *feature* es necesaria, y la posiblidad de calcular todas las *features* necesarias para ciertos modelos. Recordar que una *feature* es necesaria para una entidad e dado un modelo $\mathcal{M}$ si pertenece a todas las sufficient reasons para e dado $\mathcal{M}$.

3.2.1. Detección de necesidad

El problema de decidir si una *feature* es necesaria se puede definir de la siguiente manera:

PROBLEMA: NECESSARY$_\mathcal{L}$ ENTRADA: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$ definido sobre un conjunto de *features* $\mathcal{F}$, una entidad $e \in \texttt{ent}(\mathcal{F})$, y una *feature* $x \in \mathcal{F}$. SALIDA: x es necesaria para $\mathcal{M}(e)$ (i.e. $Nec(x, \mathcal{M}, e)$).

Una caracterización directa para las *features* necesarias es:

$$Nec(x, \mathcal{M}, e) \;\Leftrightarrow\; \forall S \in SR(\mathcal{M}, e) : \; x \in S \tag{3.2}$$

Adicionalmente, se propone una caracterización más interesante:

Teorema 42 (Caracterización de features necesarias)**.** *Sea $\mathcal{M}$ un modelo definido sobre un conjunto de features $\mathcal{F}$, y sea $e \in \texttt{ent}(\mathcal{F})$ una entidad. Entonces, una feature $x \in \mathcal{F}$ es necesaria para $\mathcal{M}(e)$ si y solo si $\mathcal{M}(e) \neq \mathcal{M}(e_{x=b})$, para algún $b \in \mathcal{D}_x$*[10]*.*

Demostración. De la definición de *feature* necesaria (ver 3.2) se desprende que x es necesaria para $\mathcal{M}(e)$ si y solo si $\mathcal{F} \setminus \{x\}$ no es una abductive explanation (es decir, una *reason* no necesariamente minimal) para $\mathcal{M}(e)$ (esto se debe a que de no ser así, existiría al menos una sufficient reason $S \subseteq \mathcal{F} \setminus \{x\}$[11]). A su vez, esto ocurre si y solo si $\mathcal{M}(e) = \mathcal{M}(e_{x=b})$ para todo $b \in \mathcal{D}_x$, puesto que $\texttt{cw}(e, \mathcal{F} \setminus \{x\}) = \{e_{x=b} : b \in \mathcal{D}_x\}$[12]. □

Por lo tanto, decidir si una *feature* es necesaria para un modelo y entidad dados es tan difícil como evaluar el modelo:

Corolario 43. *El problema NECESSARY$_\mathcal{L}$ se puede resolver con complejidad temporal $O(eval(\mathcal{M})\,|\mathcal{D}_x|)$, donde $eval(\mathcal{M})$ denota la complejidad temporal correspondiente a evaluar el modelo $\mathcal{M}$ de familia $\mathcal{L}$.*

Esto quiere decir que es fácil calcular *features* necesarias para cualquier familia de modelos que pueda ser evaluada eficientemente (siempre y cuando $|\mathcal{D}_x|$ sea pequeño[13]), y eso incluye a cualquier modelo *razonable*, como por ejemplo los clasificadores binarios booleanos en general (lo cual abarca a los circuitos D-DNNF y los diagramas de decisión

[10] Notar que b es un valor cualquiera del dominio de la *feature* x. Esto significa que, a pesar de que en esta tesis se esté trabajando con clasificadores booleanos, este resultado es válido para clasificadores no necesariamente booleanos, sin hacerle ninguna modificación.

[11]

[12] Recordar que la notación $e_{x=b}$ fue establecida en la Definición 6.

[13] Como los modelos que se trabajan en esta tesis utilizan *features* binarias, este valor es efectivamente $O(1)$.

binarios generales), o las redes neuronales. Esto es interesante dado que la tanto la intuición como los antecedentes en el área de explicabilidad basada en lógica podrían indicar que cualquier intento de aportar explicabilidad a un modelo complejo como una red neuronal sería computacionalmente inviable; sin embargo, se puede observar aquí que este no es el caso para las *features* necesarias.

Sin embargo, otra posible conclusión que se puede obtener de esto es que el problema de decidir si una *feature* es necesaria probablemente no sea tan útil a efectos de la explicabilidad como pueden serlo problemas más difíciles como $\text{RELEVANT}_{\mathcal{L}}$.

Adicionalmente, en [39] se prueba que este problema es tratable para cualquier clase de modelo para la cual sea tratable el problema de decidir si un conjunto de *features* es una abductive explanation (es decir, una sufficient reason no necesariamente minimal). Esto implica que $\text{NECESSARY}_{\mathcal{L}}$ está en P cuando $\mathcal{L}$ es FBDD, o incluso d-DNNF u otras familias de circuitos. Resumiendo, el aporte de esta tesis respecto a este tema consta de dos grandes partes:

1. La caracterización de las *features* necesarias, que resalta lo fuerte que resulta la condición de necesidad para una *feature*.

2. La extensión del resultado de [39], que solo se cumplía para ciertos modelos particulares, a prácticamente todos los modelos posibles (más precisamente, todos en los que la evaluación se pueda realizar en tiempo polinomial).

3.2.2. Necesidad en árboles de decisión

En el caso particular de los árboles de decisión, no sólo es posible decidir si una *feature* es necesaria en tiempo polinomial (como consecuencia del Corolario 43), sino que también es posible calcular todas las *features* necesarias en tiempo lineal:

Teorema 44. *Sea T un árbol de decisión booleano definido sobre un conjunto de features $\mathcal{F}$, y sea $e \in \texttt{ent}(\mathcal{F})$ una entidad. Es posible calcular todas las features necesarias para $T(e)$ en $O(|\mathcal{F}| + |T|)$ tiempo.*

Demostración. Para demostrar este resultado, se describirá un algoritmo que calcula las *features* necesarias para $T(e)$:

- Recorrer T hasta encontrar el valor de $T(e)$, definiendo $P = v_0, \ldots, v_k$ como el camino desde la raíz de T hasta alguna de sus hojas.
- Crear un vector unidimensional m de tamaño $\mathcal{F}$.
- Colocar:

$$m[x] = \begin{cases} min_{0 \leq i \leq k}(\texttt{feat}(v_i) = x) & \text{si } \exists\, i \,|\, 0 \leq i \leq k \wedge \texttt{feat}(v_i) = x, \\ \infty & \text{en caso contrario} \end{cases}$$

  Esto se puede hacer en $O(|\mathcal{F}| + |T|)$, y el resultado es un vector en donde el elemento en la posición x corresponde al primer vértice del camino P donde el camino para $e_{x=\neg e(x)}$ diverge de aquel para e.

Con la construcción descrita, se puede calcular el valor de $T(e_x)$ partiendo de $m[x]$, sin necesidad de recorrer cualquier otro vértice del camino P nuevamente. Notar además que para cada feature $x \in \mathcal{F}$, los vértices de T que son recorridos serán distintos, y por lo tanto se puede calcular todos los valores $T(e_x)$ en tiempo total $O(|T|)$. □

Esto es una mejora sustancial de complejidad frente al algoritmo presentado en [6], que tiene una complejidad temporal total de $O(|\mathcal{F}|\,|T|^2)$. En concreto, se pasa de un algoritmo cuadrático a uno lineal, lo cual implicaría una considerable disminución en el tiempo de ejecución para entradas grandes.

Por otro lado, el algoritmo descrito en esta tesis funciona únicamente para *features* con dominio booleano; esto se generaliza en [17] a *features* categóricas en general, e incluso numéricas, lo cual no resulta trivial.

3.2.3. Necesidad en diagramas de decisión binarios

El resultado probado para árboles de decisión puede ser extendido a los diagramas binarios de decisión libres (FBDDs):

Teorema 45. *Sea D un FBDD definido sobre el conjunto de features $\mathcal{F}$, y sea $e \in \texttt{ent}(\mathcal{F})$ una entidad. Es posible calcular todas las features necesarias para $D(e)$ en $O(|\mathcal{F}| + |D|)$ tiempo.*

Demostración. Utilizando un razonamiento similar al del Teorema 44: con programación dinámica es posible calcular el valor $D_v(e)$ para cada vértice $v \in D$ y en $O(|D|)$, a partir de la siguiente identidad:

$$D_v(e) = \begin{cases} b & \text{si } v \text{ es una hoja etiquetada con } b \\ D_{\texttt{left}(v)}(e) & \text{si } e(\texttt{feat}(v)) = 0 \\ D_{\texttt{right}(v)}(e) & \text{si } e(\texttt{feat}(v)) = 1 \end{cases}$$

Sea P el camino que se sigue en D al procesar la entidad e. Observar primero que una *feature* x es necesaria para $D(e)$ si y solo si:

- $x = \texttt{feat}(v)$ para algún $v \in P$.
- $D_{\texttt{left}(v)} = D_{\texttt{right}(v)} = D(e)$

Esto es ya que, por propiedad de los FBDDs (read-once property), existe un único vértice $v \in P$ tal que $x = \texttt{feat}(v)$. Además, los vértices alcanzables desde v no estarán etiquetados con x, por lo que $D(e_{x=\neg e(x)}) = D_{\texttt{left}(v)}(e)$ y $D(e_{x=\neg e(x)}) = D_{\texttt{right}(v)}(e)$.

Por lo tanto, después de un paso de pre-cómputo de los valores D_v para cada vértices del diagrama, basta con recorrer el camino P y decidir para cada *feature* si es o no necesaria en $O(1)$, resultando la complejidad total la enunciada en este resultado. □

Nuevamente se obtuvo un algoritmo con complejidad temporal lineal (óptima) en función del tamaño de entrada, lo cual resulta una mejora sobre los algoritmos cuadráticos que se pueden deducir de los resultados de [39]; dicha mejora sería especialmente notable en FBDDs de gran tamaño.

Recordar que, a su vez, los OBDDs son un caso particular de los FBDDs, con lo cual el teorema anterior se puede extender a esta familia de modelos:

Corolario 46. *Sea D un OBDD definido sobre un conjunto de features $\mathcal{F}$, y sea $e \in \texttt{ent}(\mathcal{F})$ una entidad. Es posible calcular todas las features necesarias para $D(e)$ en $O(|\mathcal{F}| + |D|)$ tiempo.*

Sin embargo, observar que en la demostración del Teorema 45 se utiliza la *read-once property* que satisfacen los FBDDs (y por consiguiente también los OBDDs). Es por eso que no es posible extender este resultado (al menos usando el mismo estilo de razonamiento) a diagramas de decisión más generales.

3.3. *Features útiles*

El último tipo de *feature* que se tratará en esta tesis es el de las *features útiles*, ya introducido anteriormente en la Definición 24. Esta categorización de *features* es interesante dado que no depende de ninguna entidad específica, sino que alude al comportamiento general de un modelo.

Otra forma de definir a estas *features* es la siguiente: una *feature* es útil para un modelo $\mathcal{M}$ definido sobre un conjunto de *features* $\mathcal{F}$ si existe un par de entidades $e, e' \in \texttt{ent}(\mathcal{F})$ tal que $\mathcal{M}(e) \neq \mathcal{M}(e')$ y $e(y) = e'(y)$ para cada *feature* $y \in \mathcal{F} \setminus \{x\}$.

3.3.1. Detección de utilidad

Es posible caracterizar las *features* útiles como aquellas que son relevantes para alguna entidad, y a la vez como aquellas que sean necesarias para alguna entidad. Además, existe una relación entre estas *features* y los *prime implicants*:

Teorema 47. *Sea $\mathcal{M}$ un modelo no constante*[14] *definido sobre un conjunto de features $\mathcal{F}$, y sea $x \in \mathcal{F}$ una feature. Son equivalente las siguientes afirmaciones:*

i) x es útil para $\mathcal{M}$.

ii) Existe una entidad $e \in \texttt{ent}(\mathcal{F})$ tal que x es necesaria para $\mathcal{M}(e)$.

iii) Existe una entidad $e \in \texttt{ent}(\mathcal{F})$ tal que x es relevante para $\mathcal{M}(e)$.

iv) Existe un prime implicant t para φ tal que $x \in Var(t)$, donde φ es la función boolena computada por $\mathcal{M}$.

Demostración. Por el Teorema 42, vale que *i)* es equivalente a *ii)*. Además, *ii)* implica *iii)* pues toda *feature* necesaria también es relevante en modelos no constantes.

Para probar que *iii)* implica *i)*, considerar una entidad $e \in \texttt{ent}(\mathcal{F})$ tal que x es relevante para $\mathcal{M}(e)$. Sea $S \in SR(\mathcal{M}, e)$ una sufficient reason tal que $x \in S$ (existe por definición de relevancia). Luego, como S es minimal, entonces $S \setminus \{x\}$ no es una *reason*, con lo cual existe una entidad $e' \in \texttt{cw}(e, S \setminus \{x\})$ tal que $\mathcal{M}(e') \neq \mathcal{M}(e)$. Notar que como S es sufficient reason, entonces $e'(x) \neq e(x)$. Por lo tanto, vale que $\mathcal{M}(e'_{x=e(x)}) \neq \mathcal{M}(e')$, entonces x es útil para $\mathcal{M}$ por definición.

Por último, se mostrará que *i)* es equivalente a *iv)*. Sea φ la función booleana computada por $\mathcal{M}$, que está definida sobre las variables $Var(\varphi) = \mathcal{F} = \{x_1, \ldots, x_n\}$. Es decir, $\mathcal{M}(e) = 1 \iff \varphi(e) = 1$, viendo a la entidad $e \in \texttt{ent}(\mathcal{F})$ como una asignación a las variables en $Var(\varphi)$. Además, dado un término t, se escribirá $t \models e$ cuando la asignación e satisface a t.

Para ver que *iv)* implica *i)*, suponer que existe un *prime implicant* t de φ tal que $x \in Var(t)$. Sea ℓ_x el literal en t correspondiente a la variable x. Sea $t' := t \setminus \{\ell_x\}$ el término que resulta de eliminar dicho literal. Como t es un *prime implicant*, entonces por minimalidad t' no es un *implicant* de φ. Luego, existe una asignación de variables (una entidad) $e \in \texttt{ent}(\mathcal{F})$ tal que $t' \models e$ pero $\varphi(e) = 0$.

[14] Esta es una condición técnica necesaria para la demostración (particularmente para probar que *ii)* implica *iii)*), a causa de la Observación 23 y el Lema 31.

Considerar ahora la entidad $e_{x=\neg e(x)}$, que satisface al literal eliminado ℓ_x (y por lo tanto satisface también a t). Se tiene entonces:

$$\varphi(e_{x=\neg e(x)}) = 1 \quad \text{y} \quad \varphi(e) = 0,$$

mientras que e y $e_{x=\neg e(x)}$ coinciden en todas las *features* en $\mathcal{F} \setminus \{x\}$. En conclusión, x es útil para $\mathcal{M}$ por definición.

Se verá ahora la dirección opuesta: *i*) implica *iv*), por el contrarrecíproco. Suponer que $x \notin Var(t)\ \forall t \in \mathtt{PI}(\varphi)$. El objetivo es probar que x no es útil. Sean las entidades (o asignaciones a $Var(\varphi)$) $e, e' \in \mathtt{ent}(\mathcal{F})$ que cumplen que:

$$e(y) = e'(y)\ \forall\, y \neq x, \qquad \varphi(e) = 1,\ \varphi(e') = 0 \text{ sin pérdida de generalidad}^{15}$$

Como $\varphi(e) = 1$, entonces existe un $t_0 \in \mathtt{PI}(\varphi)$ tal que $t_0 \models e$, dado que $\varphi \equiv \bigvee_{t \in \mathtt{PI}(\varphi)} t$ por la Proposición 3. Debido a lo supuesto, x no aparece en t_0. Luego, vale también que $t_0 \models e'$, dado que e y e' solo difieren en el valor de x. Pero como t_0 es un implicant de φ, vale que $\varphi(e') = 1$, lo cual contradice la elección de e'. Por lo tanto, no existe un par de entidades e, e' tales que, al cambiar el valor de x, no se vea modificada la salida producida por el modelo que computa φ. Es decir que x no puede ser útil. □

La intuición de la última parte de la demostración del Teorema 47 (concretamente la prueba de que *i*) implica *iv*)) radica en que si una variable (o *feature*) no aparece en ningún *prime implicant* correspondiente a un modelo, entonces cada vez que dicho modelo responda afirmativamente, significa que tiene una razón para hacerlo que no depende de la variable en cuestión. En resumen, cambiar el valor de esa variable no puede romper con todas las razones que hicieron que el resultado sea positivo, por lo que éste no puede cambiar solo por eso. Esto quiere decir que la variable es globalmente irrelevante, que es exactamente el concepto que se intenta capturar con la noción de *utilidad*.

Esta caracterización para las *features* útiles es interesante ya que reúne distintos conceptos con los que se ha trabajado durante todo el transcurso de esta tesis, mostrando que existen fuertes relaciones entre ellos.

Al igual que ocurre para las otras familias de *features* definidas previamente, se puede definir un problema de decisión asociado a la definición de *feature* útil:

Problema: USEFUL$_{\mathcal{L}}$ Entrada: Un modelo $\mathcal{M}$ de familia $\mathcal{L}$, y una *feature* $x \in \mathcal{F}$. Salida: x es útil para $\mathcal{M}$

Se verá ahora que este problema está relacionado con uno de los problemas clásicos en lenguajes de compilación y familias de modelos [26]: el de equivalencia.

Problema: EQUIV$_{\mathcal{L}}$ Entrada: Dos modelos $\mathcal{M}_1, \mathcal{M}_2$ de familia $\mathcal{L}$ Salida: $\mathcal{M}_1 \equiv \mathcal{M}_2$

La relación entre los problemas USEFUL$_{\mathcal{L}}$ y EQUIV$_{\mathcal{L}}$ viene dada por el complemento del primero:

[15] Podría haber sido al revés, i.e. $\varphi(e) = 0$ y $\varphi(e') = 1$, pero el razonamiento que sigue es el mismo en cualquier caso.

Proposición 48. *Sea $\mathcal{L}$ una familia de modelos cerrados por condicionamiento. Entonces $\overline{USEFUL_{\mathcal{L}}} \leq_P EQUIV_{\mathcal{L}}$.*

Demostración. De la Definición 24 se desprende que una *feature* x *no* es útil para un modelo $\mathcal{M}$ si y solo si $\mathcal{M} = \mathcal{M}_{x=b}$ para todo $b \in \mathcal{D}_x$. Esto equivale a afirmar que los modelos $\mathcal{M}$ y $\mathcal{M}_{x=b}$ son equivalentes sin importar el valor de b. □

Proposición 49. *Sea $\mathcal{L}$ una familia de modelos cerrados por disyunción exclusiva. Entonces $EQUIV_{\mathcal{L}} \leq_P \overline{USEFUL_{\mathcal{L}}}$.*

Demostración. Observar que $\mathcal{M}_1 \equiv \mathcal{M}_2$ si y solo si x no es útil para el modelo $\mathcal{M} \equiv (\mathcal{M}_1 \wedge x) \vee (\mathcal{M}_2 \wedge \neg x)$. □

Además, notar que el problema $\text{USEFUL}_{\mathcal{L}}$ es coNP en el caso general, dado que es posible decidir si una *feature* x no es útil *adivinando* una entidad e y un valor b en el dominio de x. Como consecuencia de esto y de las Proposiciones 48 y 49, se puede afirmar que el problema $\text{USEFUL}_{\texttt{DNF}}$ resulta coNP-completo. Además, de esto se desprende que el problema $\text{USEFUL}_{\texttt{DT}}$ es tratable (ya que los árboles de decisión están cerrados por negación y disyunción exclusiva, y decidir la equivalencia de dos árboles de decisión es posible en tiempo polinomial); y también será tratable $\text{USEFUL}_{\texttt{OBDD}}$, dado que también es una familia cerrada por disyunción exclusiva, y la existencia de una forma canónica [13] (correspondiente a la que resulta después del procedimiento de *reducción* mencionado en la Definición 15, la cual es también polinomial) permite verificar eficientemente la equivalencia entre dos OBDDs [26].

En cuanto a los FBDDs y los d-DNNFs, el problema de equivalencia de modelos no tiene una complejidad conocida al momento de redactar este trabajo [26], por lo que tampoco la tendrán los problemas $\text{USEFUL}_{\texttt{FBDD}}$ o $\text{USEFUL}_{\texttt{d-DNNF}}$.

Observación 50. *Estos resultados tienen puntos en común con el capítulo 5 de [69], según lo que indicó el propio autor como respuesta a [17]. En dicho trabajo, se plantea la utilidad desde el punto de vista del sesgo que tiene un modelo con respecto a ella. Este trabajo no era conocido al momento de realizar esta tesis, y se considera una coincidencia interesante entre dos líneas de investigación diferentes.*

3.3.2. Feature ranking basado en utilidad

Dada una *feature*, recordar que el hecho ser (o no) útil es independiente de una entidad particular del modelo. Esto permite pensar en las *features* útiles como una manera de analizar el comportamiento de un modelo, y evaluar sus *features* de acuerdo a *qué tan útiles* sean.

En este sentido, resulta conveniente contar con la caracterización dada en el Teorema 47: en particular, considerar el hecho de que una *feature* x es útil si existe alguna entidad para la cual es necesaria. Esto genera la posibilidad de construir un *puntaje* (o *score*) que se le asignaría a cada *feature* del modelo de acuerdo a su utilidad; esto es, de acuerdo a la cantidad de entidades para las cuales esa *feature* sea necesaria. A su vez, esto da lugar a un posible ordenamiento para las *features*, de acuerdo a este puntaje, de forma similar a lo propuesto por los métodos de *feature ranking* usuales.

Esta idea de puntuar las *features* se puede relacionar con aquella presentada en [11], que está basada en la noción de *causa contrafáctica* (counterfactual cause, estudiada en [37]). En el contexto de su trabajo, una causa contrafáctica para una entidad e para la

predicción de un clasificador booleano $\mathcal{M}$ tal que $\mathcal{M}(e) = 1$, es una *feature* x junto con un valor b tales que si se toma la entidad e y se fija el valor de la *feature* x en b, entonces la evaluación de $\mathcal{M}$ en la entidad resultante es 0.

La forma de puntuación propuesta en esta tesis asigna importancia a una *feature* de manera proporcional a la cantidad de entidades que admiten una causa contrafáctica basada en dicha *feature*. Este puntaje se puede calcular a través de una reducción a *model counting* (esto es: contar la cantidad de asignaciones que satisfacen una fórmula booleana, o, en este caso, a un modelo que computa una fórmula booleana[16]) para modelos cerrados por condicionamiento, negación y conjunción.

Proposición 51. *Sea $\mathcal{M} \in \mathcal{L}$ un modelo basado en un clasificador booleano definido sobre un conjunto de features $\mathcal{F}$, con $\mathcal{L}$ cerrada por condicionamiento, negación y conjunción. Sea $x \in \mathcal{F}$ una feature. Se define el valor η de la siguiente manera:*

$$\eta := |e \in \texttt{ent}(\mathcal{F}) : Nec(x, \mathcal{M}, e)|$$

Entonces, es posible calcular η con dos llamados a model counting para $\mathcal{L}$.

Demostración. Se contará la cantidad de entidades para las cuales x *no* es necesaria. Por el Teorema 42, x no es necesaria para $\mathcal{M}(e)$ si y solo si para todo $b \in \mathcal{D}_x$, vale que $k = \mathcal{M}(e) = \mathcal{M}(e_{x=b})$. Luego, la cantidad de entidades para las cuales x no es necesaria es:

$$CT(\bigwedge_{b \in \mathcal{D}_x} \mathcal{M}_{x=b}) + CT(\neg \bigwedge_{b \in \mathcal{D}_x} \mathcal{M}_{x=b})$$

, donde $CT(\mathcal{M})$ es la cantidad de entidades que acepta el modelo $\mathcal{M}$. □

En el caso particular de los árboles booleanos de decisión, se tiene como consecuencia:

Corolario 52. *Dado un árbol booleano de decisión $T \in DT$ definido sobre un conjunto de features $\mathcal{F}$, y una feature $x \in \mathcal{F}$, es posible calcular la cantidad de entidades para las cuales x es necesaria en tiempo $O(|T|^2)$.*

Demostración. Dados dos árboles binarios de decisión T_1, T_2, el modelo $T_1 \wedge T_2$ se puede representar como un árbol de tamaño $O(|T_1|\,|T_2|)$, concatenando a cada hoja de T_1 etiquetada con 1, una copia de T_2.

Por [26], el *model counting* se puede realizar en tiempo lineal para árboles de decisión. Por lo tanto, en este caso el algoritmo de la Proposición 51 se puede implementar con una complejidad como la enunciada en este resultado. □

De este resultado se concluye que el sistema de *feature ranking* basado en utilidad que se presenta en esta tesis es computable en tiempo polinomial (en particular, cuadrático) para árboles de decisión. La pregunta que surge naturalmente entonces es si esto se extiende a otros modelos. Para el caso de los FBDDs, no se puede afirmar actualmente que sea posible extender el resultado, dado que no se conoce si es una familia de modelos cerrada por conjunción: en los árboles de decisión, calcular la conjunción entre dos árboles es computacionalmente fácil (por el procedimiento de *concatenación* de árboles mencionado en la demostración del Corolario 52); pero construir la conjunción de dos FBDDs no es

[16] Las particularidades del *model counting* escapan al alcance de este trabajo; se puede conocer más al respecto en [34].

tan sencillo. Otra familia de modelos para la cual no aplican las hipótesis de la Proposición 51 son los circuitos d-DNNF, ya que a pesar de estar cerrados por condicionamiento, no están cerrados por conjunción ni negación [22][Proposición 1].

Sin embargo, existen familias de modelos entre DT y FBDD (u OBDD) para las que sí parece posible calcular eficientemente la cantidad de entidades para las cuales una *feature* es necesaria (y por tanto también sería eficiente computar el puntaje propuesto). Por ejemplo, los *multivariate decision trees* (MDTs, estudiados en el contexto de la explicabilidad en [18]) son semánticamente similares a los árboles de decisión, con la diferencia de que las decisiones (los vértices internos) están representadas como predicados multivariados (llamados *constraints* en el trabajo citado), que son preguntas que dependen de dos o más *features* de entrada. Por lo tanto, un potencial camino para extender estos resultados a esa familia de modelos sería comenzar por comprobar que está cerrada por condicionamiento, conjunción y negación. A grandes rasgos, esto parece factible, dado que: *i*) reemplazando cada *constraint* en la que aparezca una *feature* bajo la cual se esté condicionando (y dejando el resto de *constraints* inalteradas), se puede obtener el condicionamiento de un MDT; *ii*) invertir las etiquetas de las hojas produce la negación de un MDT; y *iii*) el mismo procedimiento descrito para árboles de decisión funciona para obtener la conjunción entre dos MDTs. Esto significa que la Proposición 51 aplicaría a esta familia de modelos.

En síntesis, una vía de trabajo a futuro es la de extender los resultados de tratabilidad del sistema de *feature ranking* basado en utilidad a modelos con mayor grado de *succinctness* que los árboles de decisión (como los multivariate decision trees ya mencionados, los *boosted trees* [8] o los random forests).

4. CONCLUSIONES

En esta tesis, se comenzó describiendo el panorama actual en el campo de la explicabilidad en modelos de inteligencia artificial (XAI), y más concretamente de la XAI basada en lógica. Se explicaron los conceptos elementales que la sostienen, como las nociones de lógica proposicional que llevan a definir los *prime implicants*, que a su vez son la base de las *sufficient reasons*, una de las herramientas más comunes para detectar *features* que describan el comportamiento de un modelo para una entidad de entrada determinada. Además, se definieron algunas familias de clasificadores que aparecen frecuentemente en la literatura (como los árboles de decisión, los FBDDs y los OBDDs), mostrando sus principales características e incluyendo ejemplos.

Se generalizaron y extendieron diversos resultados preexistentes en el campo de la explicabilidad basada en lógica para clasificadores booleanos. En particular, se profundizó en el cálculo de *features* relevantes y necesarias, proponiendo tanto algoritmos como resultados formales que los fundan. Por ejemplo, se aportó más detalle y sustento formal al algoritmo presentado en [6] para el cálculo de *features* relevantes en árboles de decisión booleanos, arribando a una cota de complejidad diferente a la de dicho trabajo en base a los resultados auxiliares mencionados (Teorema 34). Además, se estudió problemas relacionados a estos tipos de *features*, mostrando su carácter intratable en algunos casos, así como también su tratabilidad en otros. Para estos últimos casos, se mostró que es posible implementar algoritmos eficientes para resolverlos. Concretamente, se estudió dos generalizaciones para el problema de relevancia: *sufficient reasons* con estructura (Teorema 39) y *features* $k-$relevantes (Teorema 40 y Proposición 41). Como parte del proceso de estudio de la complejidad computacional de estos problemas, se aportó también una cota de complejidad para un problema de vertex covers en grafos (Corolario 38), que no se había encontrado en la bibliografía consultada.

Cabe destacar también que el problema de calcular *features* relevantes para OBDDs quedó abierto, mientras que el equivalente para árboles de decisión booleanos fue discutido aquí, así como su generalización a árboles de decisión con *features* categóricas y numéricas fue estudiado en el artículo [17] presentado en conjunto con esta tesis (por el momento en estado de *pre-impresión*). El caso para otros modelos con mayor *succinctness*, como los FBDDs, ya había sido analizado (y concluida su intratabilidad) por [41, 39].

Respecto al cálculo de *features* necesarias, se extendieron los resultados de [39], mostrando la eficiencia de detectar esta característica para una *feature* dada (Corolario 43) para cualquier modelo cuya evaluación sea posible en tiempo polinomial. No solo eso, sino que también se demostró que es posible calcular todas las *features* necesarias en tiempo lineal para árboles de decisión booleanos (Teorema 44) y FBDDs (Teorema 45). Una posible línea de trabajo a futuro es extender estos últimos resultados a modelos basados en circuitos booleanos, como por ejemplo los d-DNNFs.

Por otro lado, se presentó una noción de importancia para *features* independiente de la entidad que se considere (la *utilidad*), y se explicó que este concepto está vinculado con la relevancia, la necesidad y los *prime implicants* (Teorema 47). En cuanto a la complejidad de detectar la *utilidad* de una *feature*, se mostró que está vinculada con decidir la equivalencia entre dos modelos (Proposiciones 48 y 49). Esto se traduce en la intratabilidad del problema para familias de modelos de mayor *succinctness*, pero también en la tratabilidad para

modelos simples como árboles de decisión y OBDDs. Aquí se abre otra posible oportunidad de trabajo a futuro, dado que no se conoce la clase de complejidad a la que pertenece el problema de equivalencia de modelos para FBDDs y circuitos d-DNNF (y por tanto tampoco se conoce una cota de complejidad para calcular *features* útiles en estos modelos).

Por último, se propuso un sistema de puntuación basado en utilidad de *features* y a su relación con el problema de contar la cantidad de entidades para las cuales una *feature* es necesaria (Proposición 51), que podría ser usado para implementar una herramienta de *feature ranking* en el futuro. Esto sería particularmente viable para árboles de decisión, dado que el cálculo de este puntaje se puede realizar en tiempo cuadrático respecto del tamaño del árbol (Corolario 52). Surge como potencial mejora a futuro la extensión de este sistema de *ranking* a otras familias de modelos, y su consecuente evaluación en experimentos concretos (lo cual fue realizado para el caso de árboles de decisión en [17]).

BIBLIOGRAFÍA Y REFERENCIAS

[1] Akers. Binary decision diagrams. *IEEE Transactions on computers*, 100(6):509–516, 1978.

[2] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. In *Ethics of data and analytics*, pages 254–264. Auerbach Publications, 2022.

[3] Marcelo Arenas, Daniel Baez, Pablo Barceló, Jorge Pérez, and Bernardo Subercaseaux. Foundations of symbolic languages for model interpretability. *Advances in neural information processing systems*, 34:11690–11701, 2021.

[4] Marcelo Arenas, Pablo Barceló, Leopoldo Bertossi, and Mikaël Monet. On the complexity of SHAP-score-based explanations: Tractability via knowledge compilation and non-approximability results. *Journal of Machine Learning Research*, 24(63):1–58, 2023.

[5] Marcelo Arenas, Pablo Barceló, Diego Bustamante, Jose Caraball, and Bernardo Subercaseaux. A uniform language to explain decision trees. *arXiv preprint arXiv:2310.11636*, 2023.

[6] Gilles Audemard, Steve Bellart, Louenas Bounia, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. On the explanatory power of decision trees. *arXiv preprint arXiv:2108.05266*, 2021.

[7] Gilles Audemard, Frédéric Koriche, and Pierre Marquis. On tractable XAI queries based on compiled representations. In *17th International Conference on Principles of Knowledge Representation and Reasoning (KR'20)*, 2020.

[8] Gilles Audemard, Jean-Marie Lagniez, Pierre Marquis, and Nicolas Szczepanski. Computing abductive explanations for boosted trees. In *International Conference on Artificial Intelligence and Statistics*, pages 4699–4711. PMLR, 2023.

[9] Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. *Advances in neural information processing systems*, 33:15487–15498, 2020.

[10] Leopoldo Bertossi and Jorge E León. Compiling neural network classifiers into boolean circuits for efficient SHAP-score computation. *Proceedings of AMW*, 2023.

[11] Leopoldo Bertossi, Jordan Li, Maximilian Schleich, Dan Suciu, and Zografoula Vagena. Causality-based explanation of classification outcomes. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, pages 1–10, 2020.

[12] Elazar Birnbaum and Eliezer L Lozinskii. Consistent subsets of inconsistent systems: structure and behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 15(1):25–46, 2003.

[13] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[14] Randal E Bryant. Binary decision diagrams. *Handbook of model checking*, pages 191–217, 2018.

[15] Erik Brynjolfsson and Andrew McAfee. *The second machine age: Work, progress, and prosperity in a time of brilliant technologies*. WW Norton & company, 2014.

[16] Erik Brynjolfsson and Tom Mitchell. What can machine learning do? Workforce implications. *Science*, 358(6370):1530–1534, 2017.

[17] Tomás Capdevielle and Santiago Cifuentes. Feature relevancy, necessity and usefulness: Complexity and algorithms. *arXiv preprint arXiv:2505.09640*, 2025.

[18] Clément Carbonnel, Martin Cooper, and Joao Marques-Silva. Tractable explaining of multivariate decision trees. In *KR 2023-20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 127–135, 2023.

[19] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikoalei, Jérôme Monnot, and Florian Sikora. Extension of vertex cover and independent set in some classes of graphs. In *International Conference on Algorithms and Complexity*, pages 124–136. Springer, 2019.

[20] Santiago Cifuentes, Leopoldo Bertossi, Nina Pardal, Sergio Abriola, María Vanina Martínez, and Miguel Romero. The distributional uncertainty of the SHAP score in explainable machine learning. In *ECAI 2024*, pages 971–978. IOS Press, 2024.

[21] Morning Consult. IBM global AI adoption index. Enterprise report. *Available online: https://www.itransition.com/machine-learning/statistics*, 2023.

[22] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

[23] Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *AAAI/IAAI*, pages 627–634, 2002.

[24] Adnan Darwiche. Three modern roles for logic in ai. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 229–243, 2020.

[25] Adnan Darwiche et al. New advances in compiling CNF to decomposable negation normal form. In *Proc. of ECAI*, pages 328–332. Citeseer, 2004.

[26] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[27] Alexis de Colnet and Pierre Marquis. On the complexity of enumerating prime implicants from decision-DNNF circuits. *arXiv preprint arXiv:2301.13328*, 2023.

[28] Alexis De Colnet and Stefan Mengel. Lower bounds for approximate knowledge compilation. *arXiv preprint arXiv:2011.13721*, 2020.

[29] Rolf Drechsler and Detlef Sieling. Binary decision diagrams in theory and practice. *International Journal on Software Tools for Technology Transfer*, 3:112–136, 2001.

[30] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM (JACM)*, 42(1):3–42, 1995.

[31] European Commission. Coordinated plan on Artificial Intelligence. *COM (2018) 795 Final*, 2018.

[32] Daniel Fryer, Inga Strümke, and Hien Nguyen. Shapley values for feature selection: The good, the bad, and the axioms. *IEEE Access*, 9:144352–144360, 2021.

[33] Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.

[34] Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of satisfiability*, pages 993–1014. IOS press, 2021.

[35] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

[36] David Gunning and David Aha. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine*, 40:44–58, 2019.

[37] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *The British journal for the philosophy of science*, 2005.

[38] Xuanxiang Huang. Recent advances in formal explainability. 2023.

[39] Xuanxiang Huang, Martin C Cooper, Antonio Morgado, Jordi Planes, and Joao Marques-Silva. Feature necessity & relevancy in ML classifier explanations. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–186. Springer, 2023.

[40] Xuanxiang Huang, Yacine Izza, Alexey Ignatiev, Martin Cooper, Nicholas Asher, and Joao Marques-Silva. Tractable explanations for d-DNNF classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5719–5728, 2022.

[41] Xuanxiang Huang, Yacine Izza, Alexey Ignatiev, Martin C Cooper, Nicholas Asher, and Joao Marques-Silva. Efficient explanations for knowledge compilation languages. *arXiv preprint arXiv:2107.01654*, 2021.

[42] Xuanxiang Huang and Joao Marques-Silva. The inadequacy of Shapley values for explainability. *arXiv preprint arXiv:2302.08160*, 2023.

[43] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On explaining decision trees. *arXiv preprint arXiv:2010.11034*, 2020.

[44] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On tackling explanation redundancy in decision trees. *Journal of Artificial Intelligence Research*, 75:261–321, 2022.

[45] Alon Jacovi, Swabha Swayamdipta, Shauli Ravfogel, Yanai Elazar, Yejin Choi, and Yoav Goldberg. Contrastive explanations for model interpretability. *arXiv preprint arXiv:2103.01378*, 2021.

[46] Xiaoxiao Li, Yuan Zhou, Nicha C Dvornek, Yufeng Gu, Pamela Ventola, and James S Duncan. Efficient shapley explanation for features importance estimation under uncertainty. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part I 23*, pages 792–801. Springer, 2020.

[47] Mark H Liffiton and Karem A Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40:1–33, 2008.

[48] Peter Lipton. Contrastive explanation. *Royal Institute of Philosophy Supplements*, 27:247–266, 1990.

[49] Yang Lu. Artificial intelligence: a survey on evolution, models, applications and future trends. *Journal of Management Analytics*, 6(1):1–29, 2019.

[50] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[51] Tambiama André Madiega. EU guidelines on ethics in artificial intelligence: Context and implementation. *European Parliamentary Research Service*, 2019.

[52] Joao Marques-Silva. Logic-based explainability in machine learning. In *Reasoning Web. Causality, Explanations and Declarative Knowledge: 18th International Summer School 2022, Berlin, Germany, September 27–30, 2022, Tutorial Lectures*, pages 24–104. Springer, 2023.

[53] Pierre Marquis. Compile! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[54] Reda Marzouk, Shahaf Bassan, Guy Katz, and Colin de la Higuera. On the computational tractability of the (many) Shapley values. *arXiv preprint arXiv:2502.12295*, 2025.

[55] Nestor Maslej et al. The AI index 2025 annual report. *AI Index Steering Committee, Institute for Human-Centered AI, Stanford University*, 2025.

[56] John C McCallum. Price and performance changes of computer technology with time. *U.S. Bureau of Labor Statistics – with minor processing by Our World in Data. Available online: https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage*, 2023.

[57] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[58] Andrew Ng. What artificial intelligence can and can't do right now. *Harvard Business Review*, 9(11):1–4, 2016.

[59] Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Linear-time counting algorithms for independent sets in chordal graphs. In *Graph-Theoretic Concepts in Computer Science: 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers 31*, pages 433–444. Springer, 2005.

[60] Umut Oztok and Adnan Darwiche. On compiling cnf into decision-dnnf. In *International Conference on Principles and Practice of Constraint Programming*, pages 42–57. Springer, 2014.

[61] Jo Reichertz. Abduction: The logic of discovery of grounded theory. *The SAGE handbook of grounded theory*, pages 214–228, 2007.

[62] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[63] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[64] Hartley Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, USA, 1967.

[65] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[66] Zhou Shao, Ruoyan Zhao, Sha Yuan, Ming Ding, and Yongli Wang. Tracing the evolution of AI in the past decade and forecasting the emerging trends. *Expert Systems with Applications*, 209:118221, 2022.

[67] Lloyd S Shapley et al. A value for n-person games. *Princeton University Press Princeton*, 1953.

[68] Dylan Slack, Anna Hilgard, Sameer Singh, and Himabindu Lakkaraju. Reliable post hoc explanations: Modeling uncertainty in explainability. *Advances in neural information processing systems*, 34:9391–9404, 2021.

[69] Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. *Universidad de Chile*, 2020.

[70] Petroc Taylor. Amount of data created, consumed, and stored 2010-2023, with forecasts to 2028. *Statista. Available online: https://www.statista.com/statistics/871513/worldwide-data-created/*, 2024.

[71] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): Toward medical XAI. *IEEE transactions on neural networks and learning systems*, 32(11):4793–4813, 2020.

[72] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of SHAP explanations. *Journal of Artificial Intelligence Research*, 74:851–886, 2022.