Departamento de Ciencias de la Computación
e Inteligencia Artificial

Descubrimiento de Conocimiento en Grafos Multi-Relacionales

V. B. Director

Memoria presentada por
Pedro Almagro Blanco
para optar al grado de Doctor
por la Universidad de Sevilla

D. Fernando Sancho Caparrini

Sevilla, Abril de 2017.

Dedicado a mi abuelo, el ingeniero D. Pedro Almagro Ruíz,
por transmitirme enseñanzas centenarias que me permiten
distinguir mejor entre las diferentes vías a seguir.

Agradecimientos

Agradecimientos especiales a Fernando, por ser un maestro ejemplar y haber sido mi guía en estos últimos años, aportándome enseñanzas que van mucho más allá del campo académico.

A mis padres Eduardo y Blanca y al resto de mi familia, por haberme apoyado en cada una de las decisiones que he tomado en mi vida, y por haber creído en mí durante todos estos años.

A Joaquín, María del Carmen, Juan Carlos, Diego, Andrés, Juan, Gabi, Josema y Jaime, así como el resto de compañeros y colaboradores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla, por todos los aportes, consejos y ayudas ofrecidas.

A la Universidad Central de Ecuador y en concreto a todas las personas que han trabajado en el Grupo de Modelado de Sistemas Complejos, especialmente a Elizabeth por ser mi apoyo en los momentos más duros.

A Xabi, Angélica, Murdo, Juan, Nacho y el resto de amigos y familias por animarme en los momentos de flaqueo y por aguantar eternas conversaciones que han servido tanto académica como personalmente para la resolución de este trabajo.

Al laboratorio CulturePlex y a Sonia, del grupo de invetigación GESDATOS, por haberme aportado enseñanzas sin las que hubiera sido imposible llevar a cabo esta investigación.

A Kala y a Rober por su asesoramiento en las traducciones.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La inquietud del hombre por comprender el entorno en el que vive le ha permitido avanzar en su forma de enfrentarse a las incógnitas y problemas que el día a día le plantea. Esta necesidad primaria de comprender el funcionamiento de los fenómenos que le rodean le lleva a generar procedimientos de descubrimiento y explicación cada vez más eficientes sobre el funcionamiento de lo que puede percibir a través de los sentidos. Por ejemplo, la enunciación y comprensión de leyes universales de la física le permite enunciar hipótesis sobre las condiciones pasadas de los fenómenos, razonar sobre su relación con el presente que percibe, y predecir la evolución futura de los mismos, adelantándose en el proceso de toma de decisiones de eventos que todavía no han ocurrido. Sin poseer un sistema perfecto de inferencia, sin duda es la capacidad que más le diferencia del resto de seres vivos con los que comparte su existencia.

Sin embargo, de forma reiterada ha de rehacer y ampliar los mecanismos de descubrimiento que utiliza, añadiendo nuevas herramientas que le permitan seguir ampliando las fronteras del conocimiento y modificado aquellas que muestran debilidades. Entre las herramientas diseñadas en el pasado, el *Método Científico*, y los paradigmas científicos que se establecen por su aplicación, se erigen como grandes referentes, ya que se caracterizan por exigir una justificación racional de los principios que se quieren probar y por rechazar las afirmaciones absolutas a priori asumiendo que cualquiera de ellas es susceptible de ser refutada. Los paradigmas científicos más antiguos son el *empírico* (validar una hipótesis a través de repetición experimental) y el *teórico* (probar algo a través de derivaciones lógicas). Posteriormente, madura ya la era de la computación, apareció en escena un nuevo paradigma científico que se ha venido a llamar paradigma *computacional* (validar algo a través de simulaciones computacionales que complementan las observaciones experimentales). En los últimos años, y como consecuencia también del uso de sistemas informáticos para el tratamiento masivo de datos, ha surgido un nuevo paradigma de inferencia *basado en datos*,

que está dando lugar a una disciplina emergente denominada *Ciencia de los Datos* que, a pesar de no ser nueva, ha sido en las últimas décadas cuando se ha realizado un esfuerzo por dotarla de fundamentos teóricos sólidos e implementaciones viables, lo que la ha situado en el punto de mira de muchos intereses económicos, sociales y de investigación.

De entre los fenómenos que se pueden abordar con este tipo de paradigmas, son especialmente interesantes aquellos en los que tanto el análisis de los elementos como de las interacciones que se dan entre ellos son importantes, más aún cuando pueden existir diferentes tipos de interacciones o cuando hemos de considerar las propiedades internas que las caracterizan. Las estructuras de datos que permiten expresar este tipo de fenómenos, y que son necesarias para poder manipular después la información almacenada de ellos, deben estar orientadas por tanto a la descripción de sus elementos, a la de las relaciones existentes entre éstos, o presentar una orientación híbrida que permite un nivel de expresividad similar tanto para unos como para otras.

Los métodos más extendidos en Ciencia de los Datos están orientados a trabajar con estructuras que expresan de manera natural las propiedades de los elementos, pero normalmente poseen ciertas limitaciones para expresar sus interacciones de forma natural. Por ejemplo, los vectores o tablas, con los que habitualmente trabajan los algoritmos de aprendizaje automático más comunes, son muy adecuados para describir elementos, pero no así para expresar sus relaciones complejas. Paralelamente, las bases de datos clásicas proporcionan mecanismos para expresar las relaciones complejas entre los elementos, pero las tareas de consulta y modificación disponibles no están optimizadas para trabajar en base a características derivadas de estas relaciones.

La estructura matemática que parece expresar con mayor fidelidad las interacciones entre elementos de un sistema son los grafos, y recientemente han aparecido extensiones de éstos, como los *grafos con propiedades*, que permiten expresar de manera natural y simultánea todos los componentes que conforman un sistema, sus elementos y las relaciones entre éstos. Basándose en esta estructura conceptual fueron desarrolladas las *bases de datos en grafo*, que buscan almacenar la información de tal manera que tanto el proceso de almacenamiento como el posterior proceso de consulta sean eficientes cuando se realiza en base a la estructura de grafo que tienen los datos almacenados.

1.1. Grafos con Propiedades

Como hemos comentado anteriormente, los objetos matemáticos usados habitualmente para modelar las propiedades de los elementos de un sistema suelen

poseer cierta linealidad en su estructura y cuentan con buenas implementaciones computacionales (vectores, registros, tablas,...). Para aquellos casos en los que ha sido necesario disponer de herramientas de estructuración menos lineales se han creado implementaciones ad-hoc que han cubierto los requerimientos deseados (XML, RDF,...). En busca de un objeto matemático común que modele este tipo de estructuras de datos, el grafo con propiedades se destaca como un concepto que proporciona un buen equilibrio entre expresar correctamente la descripción de los elementos del sistema y la de sus relaciones. Dicha estructura, que no ha sido definida formalmente hasta hace pocos años, no posee todavía una teoría robusta que lo soporte, a pesar de que, como veremos, una correcta definición puede hacerla contener otras muchas estructuras de datos habituales.

Históricamente, los gafos uni-relacionales (aquellos en los que todas las relaciones son del mismo tipo) constituyeron las bases de la teoría matemática clásica de grafos. Los grafos multi-relacionales (aquellos en los que diferentes relaciones pueden poseer diferentes tipos) se acercan más a la descripción intuitiva de un sistema, aunque siguen mostrando algunas carencias obvias. Son los grafos con propiedades los que permiten que cada elemento (nodo o relación) posea un número indeterminado de propiedades heterogéneas asociadas. Con estas premisas, esta estructura es capaz de contener prácticamente cualquier otra estructura relacional, de tal manera que se puede adaptar al nivel de detalle del sistema que refleja en función de las necesidades de análisis posterior.

Por ello, en este trabajo haremos uso de este tipo de objetos matemáticos como estructura base, tanto en la fase previa de modelado y almacenamiento de la información como en la fase posterior de análisis e inferencia sobre ella.

1.2. Aprendizaje Automático en grafos

Una vez seleccionada una estructura matemática adecuada para describir los sistemas que nos interesan, necesitamos herramientas que nos permitan realizar inferencias de manera automática sobre éstos, normalmente con el objetivo de predecir su evolución futura o de manipularlos de manera efectiva. Tradicionalmente, los diversos paradigmas científicos han hecho uso de herramientas basadas en fundamentos lógicos, geométricos, algebraicos, analíticos, etc., y más recientemente hacen un uso extensivo de herramientas algorítmicas.

De igual forma a como, en el método científico, la enunciación de hipótesis tras la observación de fenómenos permite extraer leyes generales a partir de la experiencia, el uso de técnicas computacionales puede ser útil para obtener leyes generales a partir de ejemplos analizados con herramientas algorítmicas. Este proceso se puede hacer a diversos niveles. Podemos hacer uso de las herramientas

computacionales como medio de ayuda al proceso de inferencia del investigador, o podemos llevarlo al extremo e intentar que sea el propio algoritmo el que realice la inferencia completa y enuncie leyes de manera automática. En este último caso decimos que estamos tratando con un proceso de *aprendizaje automático* (del que hablaremos más extensamente en el capítulo 2 de Fundamentos).

En su origen, la mayoría de estas técnicas han sido ideadas para extraer patrones globales en base a una serie de ejemplos aislados, que son descritos a través de un conjunto de propiedades prefijadas, y normalmente expresados en forma de registros o tablas. Esta forma de estructurar los ejemplos coincide con el primer tipo de estructuras presentadas en la sección anterior, aquellas que permiten expresar con facilidad las propiedades de los elementos que conforman un sistema, pero no sus relaciones. Este hecho limita la capacidad de aprendizaje, ya que no considera explícitamente una parte importante del mismo. Si en el fenómeno bajo estudio las interacciones parecen ser determinantes en la comprensión del mismo, parece natural seleccionar una estructura matemática que las refleje correctamente. Sin embargo, los esfuerzos en el desarrollo del aprendizaje automático parecen haber dejado en un segundo plano este tipo de consideraciones.

Diseñar algoritmos que aprendan a partir de datos estructurados en forma de grafos con propiedades permite aprender de manera más natural tanto en base a las propiedades de los elementos de un sistema como a las relaciones existentes entre ellos. Además, aunque siempre se sacrifica información del fenómeno real, la flexibilidad del concepto de grafo permite una adaptación más fiel a la realidad, sin necesidad de realizar transformaciones adicionales que nos alejen en exceso de ella.

Si permitimos que los algoritmos de aprendizaje automático trabajen con grafos como estructura natural de la que aprender, éstos podrán manipular las relaciones de manera explícita de igual manera que lo hacen con las propiedades de los elementos. A este tipo de aprendizaje se le ha venido a llamar aprendizaje relacional (multi-relacional en caso de que existan varios tipos de relaciones entre los datos) y por suerte, y a pesar de que no ha representado un foco de atención, ha obtenido grandes avances y ha sido un área de investigación activo durante muchos años. En la literatura es habitual encontrar el aprendizaje relacional clasificado en tres grandes bloques: (1) *Statistical Relational Learning* (SRL), dentro del cual se encontrarían desarrollos como las redes lógicas de Markov [195], que utiliza una codificación de grafos multi-relacionales haciendo uso de modelos probabilísticos; (2) *Path Ranking Methods* [68, 137], que de manera explícita exploran el espacio de relaciones a través de caminos aleatorios; y (3) *Modelos Basados en Inmersiones*, que obtienen una representación vectorial del grafo a través de factorización de matrices/tensores [222, 166, 167], clusterización bayesiana [123, 224] o redes neuronales [175, 90, 37, 178]. Además

de estos tres bloques, aunque menos estudiados, podemos incluir los algoritmos que realizan descubrimiento de patrones relacionales refinando una hipótesis a través de una serie de pasos, en este último bloque podríamos incluir algoritmos como *Top-Down Induction of Logical Decision Trees* [32], *Multi-Relational Decision Tree Learning* [129] o *Graph Based Induction Decision Tree* [165], que detallaremos en el capítulo 4 de esta memoria. Aunque, como se puede observar, se han logrado avances en el aprendizaje relacional haciendo uso de árboles de decisión, redes neuronales y modelos probabilísticos, aún queda mucho camino por recorrer. Por ejemplo, hay otros algoritmos que, aún habiendo demostrado gran potencial en el aprendizaje no relacional, como es el caso de Random Forest, todavía no han sido explotados desde esta perspectiva.

Habitualmente, los modelos de aprendizaje automático se clasifican, según diversos criterios en *Supervisado* frente a *No Supervisado* (por el tipo de aprendizaje llevado a cabo), *Regesión*, *Calsificación* o *Ranking* (por el tipo de salida esperada), etc. Además, y respecto a la interpretabilidad del modelo resultante por parte de un humano, podemos clasificar los modelos en aquellos que son capaces de ofrecer una explicación que acompaña y justifica el resultado que arrojan (*modelos de caja blanca*) y aquellos que sacrifican dicha justificación en pos de una mejor eficiencia (*modelos de caja negra*).

Con respecto a los modelos de caja blanca, uno de los más representativos es el árbol de decisión, que proporciona como resultado una sucesión de tests que explican la predicción de cada uno de los ejemplos. Con respecto a los de caja negra, uno de los modelos más representativos son las redes neuronales que, a pesar de que han demostrado ser muy eficientes en tareas de clasificación y regresión, presentan grandes dificultades a la hora de ofrecer una justificación interpretable por un humano.

1.3. Objetivos de la Tesis

Ante el reducido abanico de metodologías para llevar a cabo tareas de aprendizaje automático relacional, el objetivo principal de esta investigación ha sido realizar un análisis de los métodos existentes, modificando u optimizando en la medida de lo posible algunos de ellos, y aportar nuevos métodos que proporcionen nuevas vías para abordar esta difícil tarea. Para ello, y sin nombrar objetivos relacionados con revisiones bibliográficas ni comparativas entre modelos e implementaciones, se plantean una serie de objetivos concretos a ser cubiertos:

1. **Definir estructuras flexibles y potentes** que permitan modelar fenómenos en base a los elementos que los componen y a las relaciones establecidas

entre éstos. Dichas estructuras deben poder expresar de manera natural propiedades complejas (valores continuos o categóricos, vectores, matrices, diccionarios, grafos,...) de los elementos, así como relaciones heterogéneas entre éstos que a su vez puedan poseer el mismo nivel de propiedades complejas. Además, dichas estructuras deben permitir modelar fenómenos en los que las relaciones entre los elementos no siempre se dan de forma binaria (intervienen únicamente dos elementos), sino que puedan intervenir un número cualquiera de ellos.

2. **Definir herramientas para construir, manipular y medir dichas estructuras**. Por muy potente y flexible que sea una estructura, será de poca utilidad si no se poseen las herramientas adecuadas para manipularla y estudiarla. Estas herramientas deben ser eficientes en su implementación y cubrir labores de construcción y consulta.

3. **Desarrollar nuevos algoritmos de aprendizaje automático relacional de caja negra**. En aquellas tareas en las que nuestro objetivo no es obtener modelos explicativos, podremos permitirnos utilizar modelos de caja negra, sacrificando la interpretabilidad a favor de una mayor eficiencia computacional.

4. **Desarrollar nuevos algoritmos de aprendizaje automático relacional de caja blanca**. Cuando estamos interesados en una explicación acerca del funcionamiento de los sistemas que se analizan, buscaremos modelos de aprendizaje automático de caja blanca.

5. **Mejorar las herramientas de consulta, análisis y reparación para bases de datos**. Algunas de las consultas a larga distancia en bases de datos presentan un coste computacional demasiado alto, lo que impide realizar análisis adecuados en algunos sistemas de información. Además, las bases de datos en grafo carecen de métodos que permitan normalizar o reparar los datos de manera automática o bajo la supervisión de un humano. Es interesante aproximarse al desarrollo de herramientas que lleven a cabo este tipo de tareas aumentando la eficiencia y ofreciendo una nueva capa de consulta y normalización que permita curar los datos para un almacenamiento y una recuperación más óptimos.

Todos los objetivos marcados deben ser desarrollados sobre una base formal sólida, basada normalmente en Teoría de la Información, Teoría del Aprendizaje, Teoría de Redes Neuronales Artificiales o Teoría de Grafos. Esta base debe permitir que los resultados obtenidos sean suficientemente formales como para que los aportes que se realicen puedan ser fácilmente evaluados. También se busca que los modelos abstractos desarrollados sean fácilmente implementables

sobre máquinas reales para poder verificar experimentalmente su funcionamiento y poder ofrecer a la comunidad científica soluciones útiles en un corto espacio de tiempo.

1.4. Aportaciones realizadas

El trabajo realizado ha supuesto una incursión en la formalización de grafos y el aprendizaje automático relacional y, como se refleja en esta memoria, ha permitido unificar y condensar diferentes perspectivas en dichas áreas. Además, ha permitido desarrollar nuevas técnicas para llevar a cabo este tipo de tareas haciendo uso de formalizaciones más generales y que, por tanto, permiten una mejor adaptación a diferentes contextos, así como haciendo uso de nuevos métodos de aprendizaje que son capaces de trabajar con grafos con propiedades como estructura básica de la que aprender.

Describimos a continuación las aportaciones concretas que se pueden encontrar en este trabajo:

1. **Grafo Generalizado**, estructura matemática sencilla que generaliza prácticamente todas las definiciones clásicas de grafo, desde grafo uni-relacional hasta hipergrafo con propiedades. En este trabajo se redefinen conceptos pertenecientes a la teoría de grafos desde esta nueva perspectiva y se aporta una base sólida, sencilla y flexible para soportar sistemas en los que puede resultar esencial tanto la descripción de los elementos como la de las relaciones.

2. **Extensión de las medidas**, existentes para grafos uni-relacionales, al concepto de grafo generalizado. Debido a que esta estructura generaliza la mayor parte del abanico de definiciones para grafos, esta extensión permite realizar mediciones sobre los diferentes tipos de grafo existentes.

3. **Property Query Graph**, herramienta de consulta para grafos generalizados que permite evaluar estructuras en base a su contenido y al de los elementos con los que está relacionado. Generaliza lenguajes como Cypher y otras herramientas de consulta como los Grafos de Selección. Los PQG se expresan también a través de un grafo generalizado, permitiendo así trabajar con la misma estructura tanto para la fuente de información como para la consulta.

4. **PQG-ID3**, algoritmo de aprendizaje automático relacional que permite descubrir patrones en estructuras enriquecidas de datos, y construir árboles de decisión para clasificar subgrafos a partir de un conjunto de

ejemplos clasificados. Los patrones relacionales extraídos por este algoritmo se expresan a través de un grafo generalizado, permitiendo así su fácil interpretación por parte de cualquier humano/máquina.

5. **Metodología para la inmersión de grafos generalizados en espacios vectoriales**, manteniendo la semántica original y permitiendo el descubrimiento de nueva información, recuperación de información faltante, clasificación automática de datos y aportando mejoras en otras tareas como son las consultas a larga distancia.

6. **Implementaciones** de las diversas herramientas desarrolladas a lo largo del trabajo, ofrecidas como librerías *Open Source*, y que se detallan en al apéndice de Implementación.

Además, de los puntos indicados, se han realizado aportes no menos importantes como son: un primer paso hacia una herramienta para la normalización de datos estructurados en forma de grafo a través del análisis de las estructuras vectoriales obtenidas a partir de una inmersión; una primera familia de refinamientos que permiten manipular automáticamente predicados complejos sobre grafos; y otros aportes conceptuales y técnicos de menor envergadura que no han sido nombrados en esta sección.

1.5. Estructura de la memoria

A continuación se describe el contenido de esta memoria, describiendo los diversos capítulos de que consta y que pretenden cubrir los objetivos marcados anteriormente.

En el capítulo 2, **Fundamentos**, se introducen las teorías fundamentales que sirven de eje transversal a todo el trabajo realizado y se presentan las estructuras comunes a los diferentes capítulos de esta memoria. Se presenta un marco para grafos que unifica definiciones como grafo uni-relacional o grafo con propiedades a través del concepto de grafo generalizado, redefiniendo conceptos pertenecientes a la Teoría de Grafos en un marco unificado. Además, se realiza una introducción al área del aprendizaje automático y se presentan, de manera breve, los modelos que serán utilizados a lo largo de la memoria.

En el capítulo 3, **Consulta de patrones en grafos con propiedades**, se realiza un estudio acerca de la evaluación y extracción de información en estructuras relacionales, presentando una revisión de las tecnologías y fundamentos que han abordado esta tarea. Además, se presentan los Property Query Graphs (PQG), nuestra propuesta para evaluar estructuras inmersas en un grafo con

propiedades, y que funcionan como predicados sobre subgrafos, de tal manera que son ideales como base en tareas de descubrimiento. El capítulo concluye mostrando algunos ejemplos concretos de PQG.

Posteriormente, el capítulo 4, **Árboles de decisión para grafos con propiedades**, plantea el problema de construir automáticamente árboles clasificadores de subgrafos en grafos con propiedades. Se comienza repasando el funcionamiento de los algoritmos de construcción de árboles de decisión, desde aquellos que aprenden árboles clasificadores de objetos (descritos a través de un conjunto de propiedades), hasta aquellos que aprenden a clasificar registros de una base de datos relacional teniendo en cuenta sus relaciones. La parte central del capítulo presenta el algoritmo PQG-ID3, nuestra propuesta para construir árboles de decisión multi-relacionales basada en consultas PQG. Terminamos mostrando una colección de ejemplos de árboles construidos usando esta herramienta y analizando los patrones semánticos resultantes.

A continuación, el capítulo 5, **Inmersión semántica de grafos con propiedades**, realiza una aproximación diferente al aprendizaje multi-relacional, esta vez haciendo uso de redes neuronales para aprender una codificación vectorial de un grafo con propiedades. Esta codificación permite hacer uso de los métodos de aprendizaje automático habituales diseñados para trabajar de manera natural con objetos descritos vectorialmente. Se realiza un repaso por los métodos de aprendizaje que hacen uso de redes neuronales, y se analizan métodos que han sido utilizados para realizar codificaciones de otros tipos de estructuras. Se presenta una metodología que hace uso de codificadores neuronales para llevar a cabo inmersiones de grafos con propiedades en espacios vectoriales, y se realizan experimentos sobre datos reales para verificar que la proyección obtenida permite capturar propiedades presentes en el grafo original. Además, se demuestra empíricamente que la metodología de inmersión propuesta permite llevar a cabo de manera exitosa tareas habituales de clasificación automática, extracción de información, recuperación de información faltante y consultas a larga distancia.

Por último, en **Conclusiones y trabajo futuro** presentamos las conclusiones obtenidas del proceso de investigación estructuradas de manera acorde a los diferentes bloques de esta memoria, así como las conclusiones obtenidas globalmente y de manera transversal a todo el trabajo. En este último capítulo también presentamos las líneas de trabajo futuro que se abren con esta investigación y que están relacionadas con la formalización de grafos, el aprendizaje relacional, la consulta de patrones en grafos y procedimientos de descubrimiento en bases de datos.

Capítulo 2

Fundamentos

En este capítulo presentamos los marcos teóricos que sirven de fundamento para el desarrollo del trabajo. Comenzaremos dando un marco unificado para manipular estructuras de grafos que extiende a las definiciones habituales ya que permitirá, en una sola definición, considerar los grafos e hipergrafos clásicos, así como estructuras más generales que consideran información adicional en nodos y aristas.

Como el objetivo de este trabajo es formalizar y ampliar el marco del reconocimiento de patrones en estructuras de tipo grafo, la segunda parte de este capítulo estará dedicada a introducir los fundamentos del Aprendizaje Automático, que sirve de soporte teórico e inspiración para algunas de las metodologías que daremos en capítulos posteriores. Esta sección no pretende cubrir en profundidad esta extensa área de la Inteligencia Artificial, sino únicamente hacer un poco más autocontenido el texto, destacando aquellos modelos y resultados que tienen una relación directa con los avances que presentamos.

2.1. Un marco unificado para Grafos

Desde hace ya muchos años los grafos juegan un papel fundamental en muchos campos de la ciencia, ya que suponen un nivel de abstracción que facilita la descripción, a diversos niveles, tanto de las estructuras que componen los sistemas como de las interacciones que se producen entre dichas estructuras. Así pues, podemos encontrar desde procesos económicos hasta organizaciones sociales, pasando por interacciones moleculares y relaciones semánticas entre conceptos, que hacen uso de grafos como el marco común con el que se describen y analizan de una forma unificada, y bajo la seguridad de unos resultados matemáticos robustos, dichos procesos. Todo ello gracias a que el concepto ma-

temático de grafo proporciona una herramienta lo suficientemente flexible para ser aplicada a marcos tan diversos como los mencionados, y tan potente como para ser adaptada a las necesidades diferenciadoras que estas aplicaciones demandan.

Sin embargo, la flexibilidad que lo convierte en una herramienta tan versátil, puede suponer también una debilidad cuando se intentan proporcionar herramientas transversales que permitan aplicar metodologías específicas, ya que dificulta la aceptación de un marco unificado en el que las diversas interpretaciones y variantes de un concepto tan general tengan cabida.

Como hemos comentado en la introducción, el objetivo central de esta tesis es proporcionar nuevas herramientas conceptuales y computacionales para la aplicación de metodologías de análisis a estructuras de información que vienen representadas por medio de grafos. Por ello, comenzamos este capítulo presentando una definición de grafo lo suficientemente general como para englobar a todos los tipos de grafos que vamos a necesitar para la formalización y manipulación de las diversas estructuras y metodologías que se usan a lo largo de este trabajo.

En cualquier caso, la definición que se presenta intenta abarcar las variantes más habituales que se pueden encontrar en la literatura y que nos permiten generar marcos de trabajo con una mayor capacidad de extensión.

Comenzaremos, pues, haciendo un repaso histórico acerca del concepto de *Grafo con Propiedades*, que es la antesala del concepto más general de *Grafo Generalizado* con el que trabajaremos.

2.1.1. Grafos con Propiedades

El término *Grafo con Propiedades* (*Property Graph* en inglés) hace referencia a un tipo de multi-grafo (es decir, que admite varias aristas entre los mismos nodos) dirigido en el que tanto las aristas como los nodos poseen un número indeterminado de propiedades asociadas.

El concepto de Grafo con Propiedades se ha venido utilizando desde tiempo atrás en varios contextos distintos, aunque hasta la última década no se han ofrecido las primeras definiciones formales del mismo. En [200] se da una primera formalización con el objetivo de proporcionar una definición formal de los *traversals* como herramienta fundamental de búsqueda en grafos de este tipo, orientado principalmente a su uso como soporte de las *Bases de Datos en Grafo*, mientras que en [100] se plantea una equivalencia formal entre los Grafos con Propiedades y el estándar RDF, un modelo de intercambio de datos principalmente enfocado a la Web, ampliamente usado en el contexto de las aplicaciones

orientadas a la Web Semántica, y que permite modelar de manera flexible relaciones multi-tipo entre elementos por medio de un conjunto de tripletas de la forma (*sujeto*, *predicado*, *objeto*) [246].

Habitualmente, las formalizaciones que se dan de los Grafos con Propiedades hacen uso y extienden los conceptos de Grafos unirelacionales y multirelacionales:

Definición 1. *Un* Grafo Uni-Relacional Dirigido *se define formalmente como un par* $G = (V, E)$ *donde* V *es un conjunto cualquiera que representa los* nodos *del grafo, y* $E \subseteq V \times V$ *representa el conjunto de* aristas *del grafo, un conjunto de pares ordenados de elementos de* V*.*

Esta definición de grafo uni-relacional, que prescinde completamente de estructuras enriquecidas en los conjuntos de nodos y aristas, ha sido usada durante muchos años no solo en el contexto de los resultados matemáticos de *Teoría de Grafos* sino también en entornos más aplicados, como la *Teoría de Redes Complejas* o la *Teoría de Redes Sociales*, donde ha proporcionado la herramienta conceptual básica para dar resultados generales.

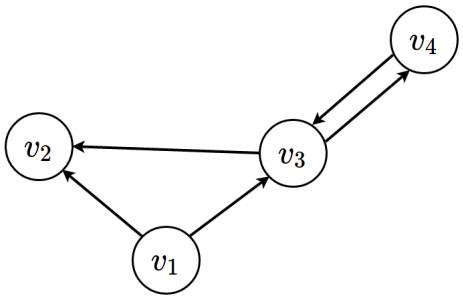


Figura 2.1: Grafo uni-relacional.

Sin embargo, en muchísimos fenómenos las relaciones existentes entre los diferentes elementos a modelar son heterogéneas, es decir, no existe un único tipo de relación entre los elementos, o precisan de estructuras de representación más ricas, por lo que la definición anterior debe ser extendida para abarcar este tipo de situaciones.

Con el objetivo de modelar situaciones en las que las relaciones entre los elementos son diversas se introduce el concepto de *Grafo Multi-relacional*, que extiende ligeramente la definición anterior añadiendo posibles etiquetados a las aristas del grafo:

Definición 2. *Un* Grafo Multi-Relacional *es una tripleta,* $G = (V, E, \tau)$*, donde* (V, E) *es un grafo uni-relacional dirigido y* $\tau : E \to \Omega$ *es una función que asigna a cada arista del grafo un* tipo de arista *de un conjunto* Ω*.*

Si enriquecemos la definición anterior agregando propiedades en forma de pares $(clave, valor)$ tanto a los nodos como a las aristas de un grafo multi-relacional dirigido obtenemos un *Grafo con Propiedades*, donde las *propiedades* permitirán representar información adicional asociada a los elementos del grafo.

Definición 3. *Un* Grafo con Propiedades *es una tupla* $G = (V, E, \tau, \mu)$ *donde* (V, E, τ) *es un grafo multi-relacional, y* μ *es una función que relaciona cada nodo o arista en el grafo con su conjunto de propiedades. Formalmente,* $\mu : (V \cup E) \times R \rightarrow S$*, donde* R *representa el conjunto de posibles* claves*, y* S *el conjunto de posibles* valores *asociados.*

Obsérvese que un Grafo con Propiedades, así definido, es un grafo binario (es decir, no proporciona una representación para estructuras más generales como los *hipergrafos*, donde las aristas pueden conectar más de dos nodos) y dirigido. Posteriormente, en la generalización que daremos en la siguiente sección, relajaremos estas condiciones.

El uso de propiedades en grafos es lo suficientemente flexible como para poder incluir la función de etiquetado de tipos de aristas, τ, como una propiedad más a partir de μ, pero en la literatura suele reflejarse por separado para enfatizar el hecho de que un Grafo con Propiedades no es más que una extensión natural de los Grafos Multi-relacionales habituales. Teniendo en cuenta que por medio de las propiedades podemos enriquecer la estructura del grafo con libertad, será común que consideremos tipos en aristas y nodos sin necesidad de explicitarlo por medio de μ.

Un ejemplo de un grafo con propiedades compuesto por dos nodos y una relación, en el que tanto los nodos como la relación tienen asociado un tipo, se muestra en la Figura 2.2.

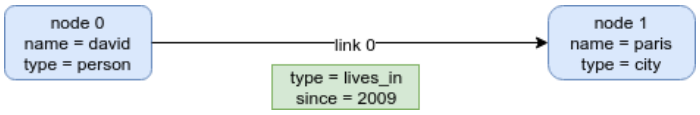


Figura 2.2: Grafo con propiedades asociadas a nodos y aristas.

Como comentamos anteriormente, uno de los objetivos de la formalización de los Grafos con Propiedades ha sido el de facilitar formas eficientes de almacenar y consultar este tipo de estructuras. Las llamadas *Bases de Datos en Grafo* hacen uso de este tipo de estructuras para implementar sistemas de almacenamiento y consultas computacionalmente eficientes cuando los accesos se realizan en base a las relaciones entre los elementos.

A diferencia de las bases de datos relacionales, en las que la información se almacena en tablas (donde las filas representan elementos, y las columnas propiedades de dichos elementos) y donde las relaciones entre los elementos

se modelan mediante columnas especiales haciendo uso de *claves foráneas*, o mediante tablas intermedias, las Bases de Datos en Grafo utilizan estructuras en memoria más similares a la estructura formal de un Grafo con Propiedades, haciendo que los nodos conectados se *apunten físicamente* entre sí de tal manera que las consultas en base a relaciones se realizan con un coste computacional mucho menor [197].

2.1.2. Grafo Generalizado

Debido a que la casuística y desarrollo que llevaremos a cabo en nuestro trabajo precisa de una definición más general de grafo que el Grafo con Propiedades presentado anteriormente, en esta sección daremos una formalización más precisa y abierta que contendrá a la anterior.

Con el fin de facilitar las definiciones que daremos a lo largo de este capítulo, y que usaremos también a lo largo de la tesis, presentamos aquí algunas notaciones y definiciones básicas.

Notación Preliminar

Dado V un conjunto cualquiera, denotaremos por:

$$V^0 = \emptyset,\ V^1 = V,\ V^{n+1} = V^n \times V,\ V^* = \bigcup_{n\geq 0} V^n$$

En general, a los elementos de V^* los llamaremos *secuencias*, *sucesiones* o *listas*. Si $x \in V^n$ entonces diremos que x tiene longitud n, y escribiremos $|x| = n$.

Si $x = (a_1, \ldots, a_n),\ y = (b_1, \ldots, b_m) \in V^*$, entonces la *concatenación* de x e y es el elemento de V^* dado por:

$$xy = (a_1, \ldots, a_n, b_1, \ldots, b_m)$$

Para cada $x = (a_1, \ldots, a_n) \in V^n$, llamaremos *conjunto soporte de* x al conjunto:

$$s(x) = \{a_i :\ 1 \leq i \leq n\},$$

y, por un abuso del lenguaje, escribiremos $a \in x$ para denotar que $a \in s(x)$.

Para cada $a \in V$, denotamos $|a|_x = \#\{i :\ x_i = a\}$ (donde $\#(A)$ denota el cardinal del conjunto A), y llamaremos *multiconjunto soporte de* x al conjunto de pares:

$$ms(x) = \{(a, |a|_x) : \ a \in x\}$$

A partir de los multiconjuntos soporte podemos definir la relación $\sim$, que se puede probar fácilmente que es de equivalencia en V^n, como:

$$x \sim y \text{ si y solo si } ms(x) = ms(y)$$

y denotaremos por $V^n_\sim = V^n/\sim$ al conjunto cociente de V^n bajo la relación de equivalencia $\sim$.

Un ejemplo de dos tuplas equivalentes es $(v_1, v_2, v_1) \sim (v_1, v_1, v_2)$, debido a que los multiconjuntos soporte de ambas tuplas son los mismos:

$$ms(v_1, v_2, v_1) = ms(v_1, v_1, v_2) = \{(v_1, 2), (v_2, 1)\}$$

Nuestra interpretación de estos conjuntos será que, así como V^n denota el conjunto de tuplas ordenadas de elementos de V, $V^n_\sim$ denota el conjunto de tuplas no ordenadas del mismo conjunto (es decir, tuplas en las que importan los elementos que aparecen, considerando las posibles repeticiones, pero no el orden en el que aparecen).

Grafo Generalizado

A continuación presentamos la definición de *Grafo Generalizado*, que abarca las diferentes variantes de grafo que necesitamos como marco común en las diversas metodologías que vamos a aplicar en nuestro trabajo.

Definición 4. *Un* Grafo Generalizado *es una tupla* $G = (V, E, \mu)$ *donde:*

- V *y* E *son conjuntos, que llamaremos, respectivamente,* conjunto de nodos *y* conjunto de aristas *del grafo.*

- μ *es una relación (habitualmente la consideraremos funcional, pero no es necesario) que asocia a cada nodo o arista en el grafo su conjunto de propiedades, es decir,* $\mu : (V \cup E) \times R \to S$*, donde* R *representa el conjunto de posibles* claves *para dichas propiedades, y* S *el conjunto de posibles* valores *asociados a las mismas.*

Habitualmente, para cada $\alpha \in R$ *y* $x \in V \cup E$*, escribiremos* $\alpha(x) = \mu(x, \alpha)$*.*

Además, exigiremos la existencia de una clave destacada para las aristas del grafo, que llamaremos incidencias *y denotaremos por γ, que asocia a cada arista del grafo una tupla, ordenada o no, de vértices del grafo, $\gamma : E \to V^* \cup V^*_{\sim}$.*

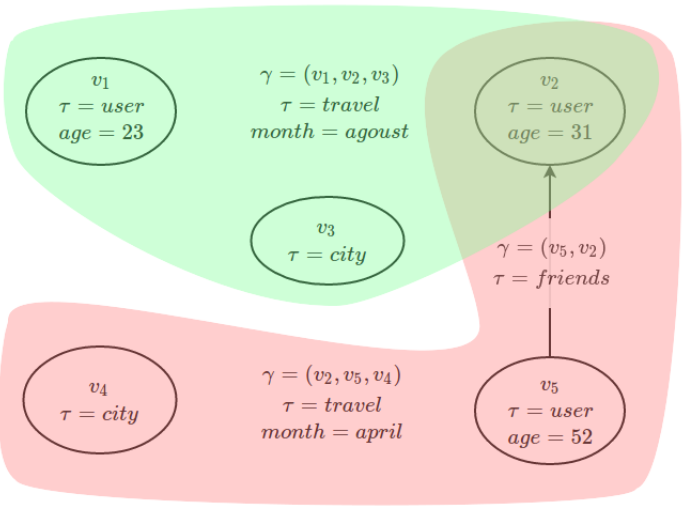


Figura 2.3: Grafo generalizado.

En general, también los denominaremos *Grafos con Propiedades*, aunque la definición que hemos presentado aquí es más general que las dadas formalmente en esos contextos y que vimos en el parágrafo anterior, y solo cuando sea necesario impondremos condiciones adicionales.

Cabe indicar que en los grafos generalizados que acabamos de mostrar, a diferencia de las definiciones tradicionales, los elementos en E son símbolos que representan a las aristas y no pares de elementos de V, y es γ la función que asocia a cada arista el conjunto de vértices que relaciona.

Gracias a que la longitud de la lista almacenada en la propiedad γ no tiene restricción en su tamaño se abre la posibilidad de que una arista conecte un número de nodos en V superior a 2, permitiendo así trabajar con hipergrafos de manera natural. En función de si dicha lista es ordenada o no, hablaremos de un grafo dirigido o no dirigido.

En algunas definiciones de hipergrafo se dice que una hiperarista es un multiconjunto de nodos dotados de una relación de orden conexa, no vacía y transitiva, que será la que establezca el orden interno entre los nodos que participan en las hiperaristas. Si el orden es total y simétrico, entonces la hiperarista se dice no dirigida. Si es total y lineal, se dice dirigida. La forma que presentamos aquí es más general, ya que permite hiperaristas parcialmente dirigidas, y pasa a ser una característica de la propia arista, y no del grafo completo.

Definición 5 (Notación y definiciones)**.** *En el contexto de las definiciones an-*

teriores, usaremos la siguiente notación y nomenclatura:

- *Habitualmente identificaremos e con $\gamma(e)$, de forma que si $v \in V$ escribiremos $v \in e$ para denotar que $v \in \gamma(e)$. Así, interpretamos la arista como la colección de nodos que conecta, tal y como siguen las definiciones más clásicas de grafos.*

- *De forma simétrica, para cada $u \in V$ escribiremos*

$$\gamma(u) = \{e \in E : \ u \in e\}$$

- *En general, un grafo generalizado puede tener combinación de aristas dirigidas y no dirigidas. Si $\gamma : E \to V^*$ diremos que el grafo es* Dirigido. *Si $\gamma : E \to V^*_\sim$ diremos que el grafo es* No Dirigido.

- *Para cada $e \in E$, se define la* aridad *de e como $\sum_{a \in e} |a|_e$.*

- *Si $\gamma : E \to V^2 \cup V^2_\sim$ diremos que el grafo es* Binario *(y coincide con la estructura de grafo más habitual). En caso contrario, diremos que el grafo es un* Hipergrafo.

- *Una arista, $e \in E$, se dice que es un* lazo *si conecta un nodo con él mismo, es decir, si tiene aridad distinta a 2 pero $s(e)$ es unitario.*

- *Una arista, $e \in E$, se dice* incidente *en un nodo, $v \in V$, si $v \in e$.*

- *Dos nodos distintos, $u, v \in V$ se dicen* adyacentes, *o* vecinos, *en G si existe $e \in E$ tal que $\{u, v\} \subseteq e$.*

- *Si existen aristas distintas en E con la misma incidencia, es decir, aristas que conectan los mismos nodos, diremos que el grafo es un* Multi-grafo.

- *Si e es una arista binaria dirigida que conecta u con v, $e = (u, v)$, escribiremos $u \xrightarrow{e} v$, y también notaremos $e^o = u$ (*output *de e) y $e^i = v$ (*input *de e). En este caso, para cada $u \in V$ escribiremos:*

$$\gamma^o(u) = \{e \in \gamma(u) : \ e^o = u\}$$

$$\gamma^i(u) = \{e \in \gamma(u) : \ e^i = u\}$$

que denotan, respectivamente, el conjunto de aristas salientes *de u y el conjunto de* aristas entrantes *en u.*

- *Dado $u \in V$, definimos el* entorno *de u en G como el conjunto de nodos, incluyendo a u, que están conectados con él, es decir:*

$$\mathcal{N}(u) = \bigcup_{e \in \gamma(u)} \gamma(e)$$

Cuando sea necesario hablaremos de entorno reducido *de u como $\mathcal{N}^*(u) = \mathcal{N}(u) \setminus \{u\}$.*

- *Cuando en G destacamos una propiedad adicional, τ, que denominaremos* tipo*, hablaremos de grafo* tipado*, que puede asignar tipos a nodos, y aristas. Si además $\tau(E)$ tiene un solo elemento (todas las aristas tienen el mismo tipo) diremos que G es* Uni-Relacional*. En caso contrario diremos que G es* Multi-Relacional*. Por defecto, entenderemos que un grafo en el que no se ha definido una propiedad* tipo *en las aristas es un grafo Uni-relacional.*

Podemos dar algunos ejemplos de las definiciones anteriores haciendo uso del ejemplo de grafo generalizado presentado en la Figura 2.3, donde podemos encontrar una arista de aridad 2 (binaria) y dos aristas de aridad 3 (ternarias), por lo que corresponde a un hipergrafo en el que no existen lazos. Dos ejemplos de entornos serían $\mathcal{N}(a) = \{a, b, c\}$ y $\mathcal{N}(b) = \{a, b, c, d, e\}$.

La noción de subgrafo se obtiene de la definición habitual añadiendo a las condiciones habituales de contención de nodos y aristas la condición de que las propiedades también se mantengan en los elementos comunes.

Definición 6. *Un subgrafo de un grafo $G = (V, E, \mu)$ es un grafo $G_S = (V_S, E_S, \mu_S)$ tal que $V_S \subseteq V$ y $E_S \subseteq E$ y $\mu_S = \mu_{|V_S \cup E_S}$. Notaremos $S \subseteq G$.*

Cuando sea necesario, podremos dejar fuera de esta igualdad algunas propiedades específicas si así se explicita.

Conectividad

Un concepto fundamental al trabajar con grafos es el de *camino*, que permite estudiar relaciones de distancia y condiciones de conectividad entre diferentes elementos del grafo, extendiendo la conectividad de las aristas a situaciones más generales.

Debido a que nuestros grafos son considerablemente más generales que los habituales (hasta el punto de contener el concepto de hipergrafo, que generalmente no se cubre en la Teoría de Grafos clásica) hemos de dar previamente algunas nociones que permitan hablar de la posición de orden que ocupa un nodo en una arista:

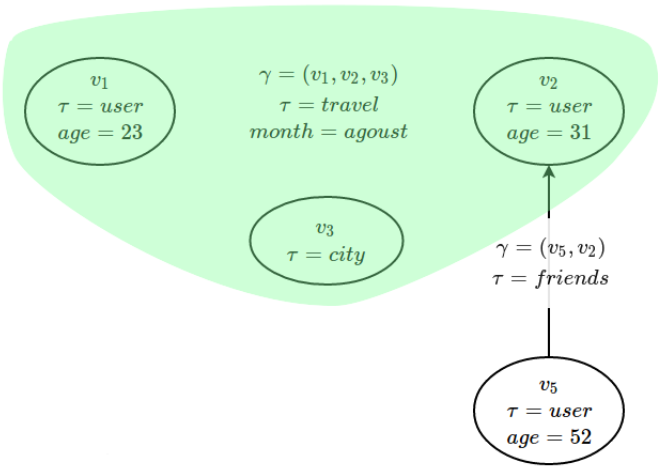


Figura 2.4: Subgrafo del grafo presentado en la Figura 2.3.

Definición 7. *Si $e \in E$ y $\gamma(e) = (v_1, \ldots, v_n) \in V^n$, entonces para cada $v_i \in s(e)$ definimos su* orden *en e como $ord_e(v_i) = i$. Si $e \in V^n_\sim$, entonces para cada $v \in s(e)$ definimos $ord_e(v) = 0$.*

Este orden define de forma natural un orden entre los nodos incidentes en una arista, y escribiremos $u \leq_e v$ para indicar que $ord_e(u) \leq ord_e(v)$.

A partir de esta relación de orden entre los nodos que conecta una arista, podemos definir de manera general qué entendemos por un camino dentro de un grafo.

Definición 8. *Dado un grafo $G = (V, E, \mu)$, el conjunto de* caminos *en G, que denotaremos por $\mathcal{P}_G$, se define como el menor conjunto verificando las siguientes condiciones:*

1. *Si $e \in E$, $u,\ v \in e$ con $u \leq_e v$, entonces $\rho = u \overset{e}{\to} v \in \mathcal{P}_G$, y $sop_V(\rho) = (u, v)$, $sop_E(\rho) = (e)$.*
   Diremos que ρ une *(o* conecta*) los vértices u y v de G, o que v* es accesible desde u por medio de ρ*, y lo notaremos por $u \overset{\rho}{\rightsquigarrow} v$.*

2. *Si $\rho_1,\ \rho_2 \in \mathcal{P}_G$, con $u \overset{\rho_1}{\rightsquigarrow} v$, $v \overset{\rho_2}{\rightsquigarrow} w$, entonces $\rho_1 \cdot \rho_2 \in \mathcal{P}_G$, con $u \overset{\rho_1 \cdot \rho_2}{\rightsquigarrow} w$, $sop_V(\rho_1 \cdot \rho_2) = sop_V(\rho_1) sop_V(\rho_2)$, $sop_E(\rho_1 \cdot \rho_2) = sop_E(\rho_1) sop_E(\rho_2)$ (la concatenación de las listas).*

En caso de que $u = v$ diremos que ρ es un camino cerrado*, y si además no se repiten aristas en ρ diremos que es un* ciclo*.*

Si $\rho \in \mathcal{P}_G$, con $sop_V(\rho) = (u_1, \ldots, u_{n+1})$ y $sop_E(\rho) = (e_1 \ldots, e_n)$, entonces escribiremos:

$$\rho = u_1 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_n} u_{n+1}$$

En general, y como no hay confusión, escribiremos $u \in \rho$ para expresar que $u \in sop_V(\rho)$, y $e \in \rho$ para expresar que $e \in sop_E(\rho)$.

En la Figura 2.5 se muestra un grafo en el que han sido marcados dos caminos, $v_1 \overset{\rho_1}{\rightsquigarrow} v_7$ (en verde) y $v_7 \overset{\rho_2}{\rightsquigarrow} v_1$ (en rojo). El camino $\rho_1 \cdot \rho_2$, resultante de la concatenación de ambos, representa un ciclo.

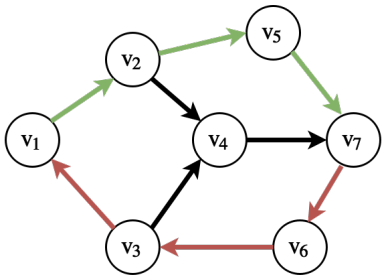


Figura 2.5: Caminos en un grafo.

En $\mathcal{P}(G)$ se puede considerar una relación de equivalencia que hace que la operación de concatenación de caminos, $\rho_1 \cdot \rho_2$, sea asociativa, de forma que

$$\rho_1 \cdot (\rho_2 \cdot \rho_3) = (\rho_1 \cdot \rho_2) \cdot \rho_3$$

A partir de esta definición recursiva es sencillo definir qué entendemos por longitud de un camino:

Definición 9. *Se define la* longitud *de un camino $\rho \in \mathcal{P}_G$, y la notaremos por $|\rho|$, como:*

1. $|(u, e, v)| = 1$
2. $|\rho_1 \cdot \rho_2| = |\rho_1| + |\rho_2|$

En caso de que estemos trabajando con grafos binarios (dirigidos o no) esta definición extiende a la definición habitual de camino, y que podemos ver escrita como (usando la notación presentada anteriormente):

Definición 10. *Un* camino *de longitud n en un grafo $G = (V, E, \mu)$ es una sucesión de aristas $(e_1, \ldots, e_n)$ tal que para cada $2 \leq i \leq n-1$ se tiene que existen $u \in e_{i-1} \cap e_i$, $v \in e_i \cap e_{i+1}$ con $u \leq_{e_i} v$.*

Diremos que el camino conecta *los vértices u y v de G si $u \in e_1$, $v \in e_n$, y además u y v mantienen el orden en sus aristas, es decir, existen $u^+ \in e_1 \cap e_2$ y $v^- \in e_{n-1} \cap e_n$ tales que $u \leq_{e_1} u^+$ y $v^- \leq_{e_n} v$.*

Nota.

- *Siguiendo una notación similar al caso de las aristas binarias dirigidas, si $\rho \in \mathcal{P}(G)$ y $u \overset{\rho}{\rightsquigarrow} v$, entonces escribiremos $\rho^o = u$ y $\rho^i = v$.*
- *Y de forma general, notaremos, respectivamente, los caminos que pasan por u, que comienzan en u, y que acaban en u, por:*

$$\mathcal{P}_u(G) = \{\rho \in \mathcal{P}(G) : \ u \in \rho\}$$
$$\mathcal{P}^o_u(G) = \{\rho \in \mathcal{P}(G) : \ \rho^o = u\}$$
$$\mathcal{P}^i_u(G) = \{\rho \in \mathcal{P}(G) : \ \rho^i = u\}$$

A partir del concepto de accesibilidad que proporciona la existencia de caminos, damos la definición habitual de grafo conexo:

Definición 11. *Un grafo G se dice* conexo *si todo par de vértices suyos puede conectarse por medio de un camino en G. La componente conexa asociada a un vértice es el conjunto de nodos que son accesibles desde él (y se pueden definir de forma trivial las componentes* entrantes *y* salientes*).*

Obsérvese que la relación *ser accesible* no es una relación de equivalencia ya que, aunque en general suele ser reflexiva y es transitiva, la simetría solo se puede asegurar si el grafo es no dirigido. En consecuencia, las componentes conexas en general no constituyen una partición en forma de clases de V.

A continuación presentamos la definición de árbol, un caso especial de grafo con importantes aplicaciones en numerosas áreas incluyendo el aprendizaje automático.

Definición 12. *Un* árbol *es un grafo binario, uni-relacional, no dirigido, conexo, y sin ciclos.*

Un árbol se dice *enraizado* si uno de sus nodos ha sido designado como *raíz*. En este caso, a pesar de que el grafo original (árbol) no es dirigido, el árbol enraizado recibe una dirección natural en sus aristas desde la raíz hacia el resto de los nodos del árbol. Con esta nueva orientación, si una arista conecta el nodo u con el nodo u diremos que v es el *hijo* de u y que u es el *padre* de v. Sólo puede haber un único nodo sin padre, que es la raíz del árbol. Los nodos que no tienen hijos se conocen como *hojas*, y los nodos que no son ni hojas ni la raíz se dicen *interiores*.

Si G es un árbol, dos vértices cualquiera de G están conectados por un único camino simple.

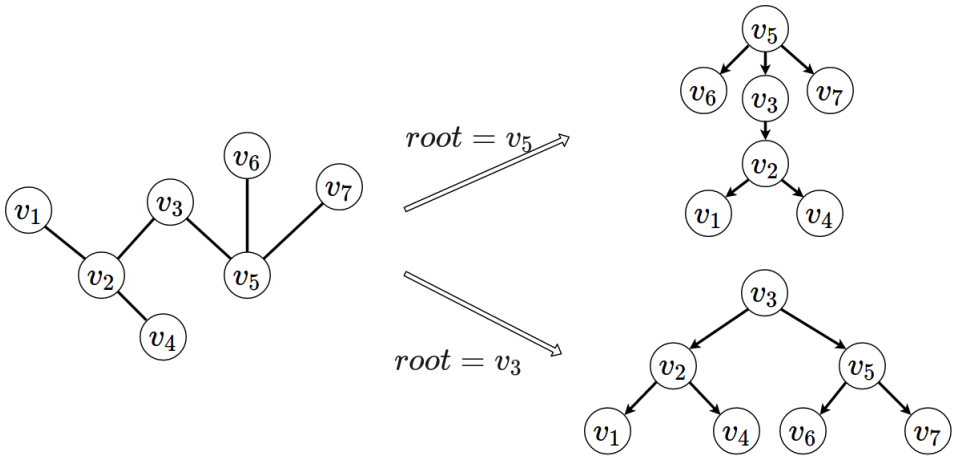


Figura 2.6: Árbol enraizado a través de diferentes nodos.

Definición 13. *Un* bosque *es un grafo en el que todas sus componentes conexas son árboles.*

Proyecciones e Isomorfismos

Adaptamos las definiciones habituales de proyección e isomorfismo al contexto de los grafos generalizados que hemos definido. Estos conceptos serán imprescindibles en algunas secciones posteriores de la tesis, y también para comprender las diferencias que presentan las soluciones que aportamos con respecto a las existentes.

En una proyección solo nos preocupamos por las propiedades estructurales dadas por nodos y aristas, y no interviene la parte semántica que proporciona μ (salvo γ):

Definición 14. *Dados dos grafos con propiedades* $G_1 = (V_1, E_1, \mu_1)$ *y* $G_2 = (V_2, E_2, \mu_2)$*, diremos que* $\pi = (f_V, f_E)$ *es una* proyección *si* $f_V : V_1 \to V_2$*,* $f_E : E_1 \to E_2$*, son funciones totales verificando:*

$$\forall v \in V_1,\ \forall e \in E_1 (v \in e \to f_V(v) \in f_E(e))$$

En los isomorfismos no solo exigimos una biyección que se comporta como una proyección (en ambas direcciones) entre las estructuras de los grafos, sino que además proyectamos propiedades (claves y valores) y pedimos que se mantenga la coherencia semántica de las mismas:

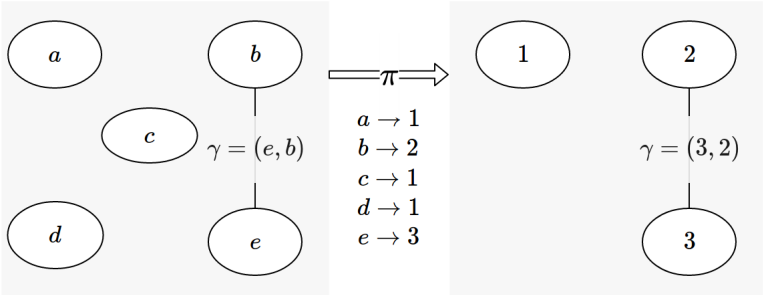


Figura 2.7: Proyección entre grafos.

Definición 15. *Dos grafos con propiedades* $G_1 = (V_1, E_1, \mu_1)$ *y* $G_2 = (V_2, E_2, \mu_2)$ *se dice que son isomorfos si existen aplicaciones biyectivas* $\pi = (f_V, f_E, f_R, f_S)$*, donde* $f_V : V_1 \to V_2$*,* $f_E : E_1 \to E_2$*,* $f_R : R_1 \to R_2$ *y* $f_S : S_1 \to S_2$*, tales que:*

1. *Si* γ_1 *y* γ_2 *son las propiedades de incidencia en* G_1 *y* G_2*, respectivamente, entonces* $f_R(\gamma_1) = \gamma_2$*.*
2. $\forall v \in V_1,\ \forall \alpha \in R_1$*, se tiene que* $f_S(\mu_1(v, \alpha)) = \mu_2(f_V(v), f_R(\alpha))$*.*
3. $\forall e \in E_1,\ \forall \alpha \in R_1$*, se tiene que* $f_S(\mu_1(e, \alpha)) = \mu_2(f_E(e), f_R(\alpha))$*.*

Las igualdades anteriores deben entenderse de tal manera que si uno de los dos términos existe, el otro también y son iguales. En particular, el isomorfismo mantiene las relaciones de incidencia.

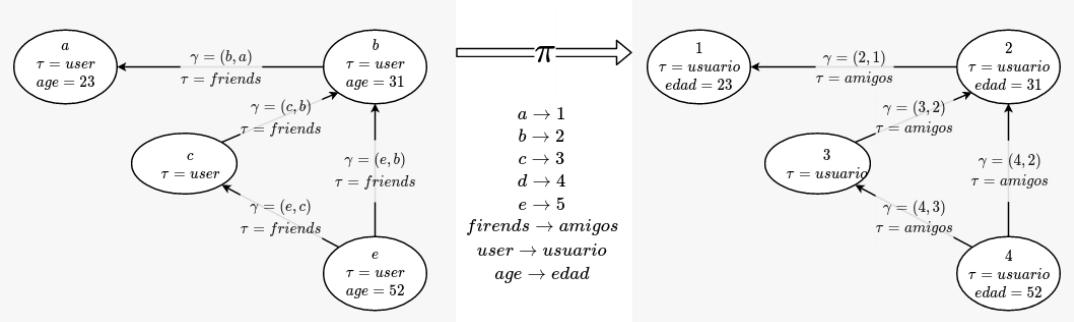


Figura 2.8: Isomorfismo entre grafos.

Obsérvese que, en particular, todo isomorfismo es una proyección biyectiva cuya inversa también es una proyección.

Teniendo en cuenta la notación que introdujimos anteriormente, y que nos permite ver las propiedades como funciones, $\alpha(x) = \mu(x, \alpha)$, y si consideramos que $f_G = f_V \cup f_E$ está bien definida y es una biyección que mantiene nodos y aristas, podemos expresar las relaciones anteriores como:

$$\forall x \in V_1 \cup E_1,\ \forall \alpha \in R_1 \text{ se tiene que } f_S(\alpha(x)) = f_R(\alpha)(f_G(x))$$

O lo que es lo mismo:

$$\forall \alpha \in R_1\ (f_S \circ \alpha = (f_R(\alpha)) \circ f_G)$$

Viendo f_R como una elevación de funciones definidas en $V_1 \cup E_1$ en funciones definidas en $V_2 \cup E_2$, podemos entonces decir que $\pi = (f_V, f_E, f_S, f_R)$ es un isomorfismo si, además de ser una colección de funciones biyectivas, f_R eleva las propiedades de G_1 a propiedades de G_2 de forma consistente con f_S, haciendo el siguiente diagrama conmutativo:

$$\begin{array}{ccc} V_1 \cup E_1 & \xrightarrow{f_G} & V_2 \cup E_2 \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle f_R(\alpha)} \\ S_1 & \xrightarrow[f_S]{} & S_2 \end{array}$$

2.2. Medidas para Grafos Generalizados

La Teoría de Grafos (y su denominación como Teoría de Redes cuando se aplica a áreas específicas fuera de la matemática, quizás cambiando algunos de sus objetivos fundamentales) ha sido una valiosa herramienta para describir y analizar sistemas que ocupan rangos tan variados como los sociales, biológicos, físicos, o de sistemas de información. Como ya hemos comentado, la mayoría de los estudios aplicados que hacen uso de esta herramienta usan grafos unirelacionales. A pesar de que este tipo de grafo ha dado buenos resultados en multitud de aplicaciones, se hace necesario extender tanto las definiciones básicas como el aparato analítico desarrollado para poder tratar casos más complejos.

Los primeros avances en métodos de análisis de redes se obtuvieron en el campo de la sociología [85] con medidas como la centralidad de los nodos (basada en el grado de los nodos y en la existencia de caminos mínimos). Con el incremento en la capacidad de procesamiento de las computadoras y la acumulación de grandes conjuntos de datos se han hecho necesarios métodos analíticos que permitan obtener resultados más potentes y específicos en datos altamente relacionados. De esta forma, se han desarrollado medidas de agrupamiento

(clustering) que detectan grupos densamente conectados, medidas que analizan procesos de difusión a través de la red, o medidas basadas en entropía o complejidad.

Junto a la respuesta al incremento de tamaño de los grafos analizados, desde hace algunos años también se han desarrollado herramientas teóricas y aplicadas con el fin de modelar redes en las que no todos los nodos, ni aristas, representan el mismo tipo de información. Junto a las soluciones que modelan conjuntos heterogéneos de nodos y aristas por medio de lo que se llaman *Grafos Multi-relacionales* [198] encontramos otras aproximaciones más específicas pero muy extendidas en determinadas áreas, como los denominados *Grafos Multi-capa* [63], donde existen diferentes *capas* y cada una de ellas modela un grafo uni-relacional, pero con posibilidad de añadir aristas entre nodos de distintas capas. Además, se han desarrollado también lenguajes de consulta y generación de grafos multi-relacionales que han intentado dar aproximaciones más abstractas [201][103].

De forma paralela, la aplicación de técnicas analíticas a redes multi-relacionales ha despertado gran interés en los últimos años debido en parte a la facilidad de uso y accesibilidad a nuevas formas de almacenamiento computacional. Un posible método a la hora de aplicar análisis tradicional, de grafos uni-relacionales, a grafos multi-relacionales es proyectar estos últimos a un grafo uni-relacional para así aplicar las medidas habituales. La forma básica de considerar esta proyección es ignorando los diferentes tipos existentes en la red original, pero también puede ser llevada a cabo a través de patrones predefinidos que, al ser detectados en la red, forman una nueva arista en la proyección [202]. También se han desarrollado técnicas analíticas para redes multi-capa, en [127] se estudian medidas relacionadas con el grado, coeficiente de *clustering*, comunidades y medidas que relacionan las capas entre sí. Algunos otros ejemplos de análisis multi-relacional podemos encontrarlos en [174], donde se analiza cómo dar medidas de agrupamiento para el caso particular de las redes bipartitas (o *Redes Two Mode*), [24], donde se dan procedimientos de descubrimiento de nuevas relaciones (*link discovery*) a partir de la asortatividad de los nodos, o [102], donde se extiende la medida *Page Rank* para que sea sensible a determinados tópicos en la red.

A continuación presentaremos algunas medidas sobre grafos generalizados, la mayoría de las cuales han sido propuestas originalmente para grafos unirelacionales y de las que ofrecemos posibles extensiones. En general, las posibilidades de definición explotan al considerar las posibles divisiones que se pueden hacer usando las propiedades de los elementos del grafo. El valor añadido que puedan aportar unas u otras está claramente asociado al dominio de aplicación concreto sobre el que se aplique. En consecuencia, las extensiones que presentamos deben ser consideradas como una muestra de las posibilidades que se ofrecen al

trabajar con definiciones más generales, y no como una presentación exhaustiva con pretensiones de fijar líneas de ningún tipo.

En lo que sigue consideraremos un grafo generalizado prefijado $G = (V, E, \mu)$ en las condiciones (y notación) que hemos visto en este mismo capítulo.

2.2.1. Centralidad

Las medidas de centralidad persiguen localizar los elementos más importantes (respecto de alguna característica comparable) de un conjunto [51]. Cuando se aplican a grafos, las medidas de centralidad suelen dar una clasificación de los nodos, aunque hay excepciones que han trabajado sobre aristas.

Grado

Una de las medidas más clásicas a la hora de clasificar los nodos de un grafo viene dada por su *grado*. El grado de un nodo en un grafo uni-relacional no dirigido es el número de aristas incidentes a dicho nodo. En el caso de los grafos uni-relacionales dirigidos se pueden añadir de forma natural dos variantes a esta medida distinguiendo entre dos tipos de grado, el *grado de entrada* y el *grado de salida*.

En el caso de lo grafos multirelacionales existen diversas extensiones que pueden dar información interesante para clasificar la centralidad del nodo.

Definición 16. *Dado $u \in V$, $\alpha \in R$ y $s \in S$, definimos el* grado de u por aristas respecto de $\alpha = s$ *como:*

$$gr^E_{\alpha=s}(u) = \#\{e \in \gamma(u) : \ \alpha(e) = s\}$$

Y el grado por nodos de u respecto de $\alpha = s$ *como:*

$$gr^V_{\alpha=s}(u) = \#\{v \in \mathcal{N}^*(u) : \ \alpha(v) = s\}$$

Dada una distribución de probabilidad en S, $\mathbb{P}$, podemos considerar los grados medios de las medidas anteriores como:

$$\overline{gr}^E_\alpha(u) = \mathbb{E}_{\mathbb{P}}[gr^E_{\alpha=s}(u)]$$

$$\overline{gr}^V_\alpha(u) = \mathbb{E}_{\mathbb{P}}[gr^V_{\alpha=s}(u)]$$

Si consideramos en S un orden prefijado (por ejemplo, cuando $S \subseteq \mathbb{N}$), podemos tratar el vector grado *dado por (si no, obtenemos una función que depende de $s \in S$):*

$$GR^E_\alpha(u) = (gr^E_{\alpha=s}(u))_{s\in S}$$
$$GR^V_\alpha(u) = (gr^V_{\alpha=s}(u))_{s\in S}$$

Obsérvese que los grados por nodo y por arista no coinciden en general, ya que en el caso de grafos generalizados puede haber más de una arista conectando dos mismos nodos (en el caso de multi-grafos).

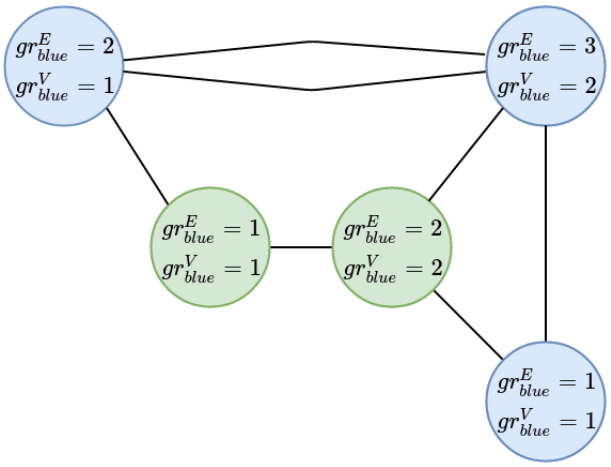


Figura 2.9: Grados por aristas y por nodos.

Uno de los casos más habituales en los que encontraremos aplicaciones para esta generalización del grado es para separar los grados en función de los tipos (τ) de nodos y aristas.

También podemos generalizar las definiciones anteriores considerando un predicado θ que haga uso de las propiedades definidas por μ de forma que podemos considerar una combinación cualquiera de las restricciones y condiciones impuestas en el conteo.

Centralidad Betweenness

Una forma habitual de centralidad está basada en la idea de que la importancia de un nodo en un grafo está relacionada con la posición que el nodo ocupa con respecto a los caminos en el grafo [85]. Si pensamos en esos caminos como vías de comunicación, los nodos que estén ubicados en más caminos serán más importantes en la comunicación. Formalmente, este concepto se define como:

Definición 17. *Dado $u \in V$, definimos la* Centralidad Betweenness *de u en G como:*

$$bc(u) = \sum_{s,t \in V, s \neq u \neq t} \frac{\sigma(s,t|u)}{\sigma(s,t)}$$

donde $\sigma(s,t|u)$ es el número total de caminos mínimos entre s y t que pasan por u, y $\sigma(s,t)$ es el número total de caminos mínimos entre s y t (independientemente de que pasen por u o no).

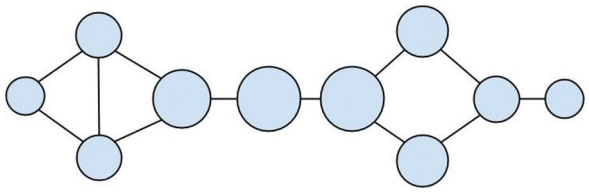


Figura 2.10: Centralidad Betweenness.

En el caso de lo grafos generalizados de nuevo se nos presentan varias posibilidades a la hora de extender la definición anterior. La primera y más general es imponiendo restricciones que han de verificar los caminos que vamos a contar (por ejemplo, por propiedades específicas en los nodos y aristas que los forman). Además, es fácil extender esta misma idea para que se pueda definir la centralidad betweenness de aristas también. Así pues:

Definición 18. *Dado θ predicado definido en $\mathcal{P}(G)$ a partir de propiedades de μ, y $x \in V \cup E$, se define la Centralidad Betweenness de x respecto de θ, como:*

$$bc_\theta(x) = \sum_{s,t \in V, s \neq x \neq t} \frac{\sigma_\theta(s,t|x)}{\sigma_\theta(s,t)}$$

donde $\sigma_\theta(s,t|x)$ es el número total de caminos mínimos entre s y t que pasan por x y verifican θ, y $\sigma_\theta(s,t)$ es el número total de caminos mínimos entre s y t que verifican θ (independientemente de que pasen por x o no).

Obsérvese que la definición clásica unirelacional es equivalente a la generalizada cuando θ es una tautología. Además, la definición que hemos dado es igualmente válida para nodos y aristas.

Page Rank

Las medidas de centralidad basadas en caminos mínimos son muy útiles a la hora de clasificar elementos en un grafo debido en gran medida a que

tienen en cuenta la estructura *completa* de la red pero tienen el inconveniente de que su cálculo exacto consume muchos recursos computacionales, lo que puede hacer que sean intratables para redes grandes. Una excelente alternativa son las medidas de centralidad basadas en autovectores [168], que miden la importancia de un nodo en una red en función de la conectividad de éste a otros nodos que también están bien conectados según este mismo criterio: un nodo es *importante* si está conectado a otros nodos importantes.

Aunque el cálculo de este tipo de medidas puede parecer computacionalmente costoso, debido a que se fundamentan en un proceso recursivo, se obtienen muy buenas aproximaciones haciendo uso de caminos aleatorios [170]. Vamos a explicar la idea a través del modelo clásico de grafo uni-relacional dirigido:

Supongamos un caminante que se mueve de manera aleatoria de nodo en nodo a través de las aristas de un grafo uni-relacional dirigido. Para facilitar la explicación, supongamos que los nodos del grafo son $V = \{v_1 \ldots, v_n\}$.

Si denotamos por p_i^t la probabilidad de que el caminante se encuentre en el nodo v_i del grafo en el instante t, y partimos de que originalmente el caminante puede empezar con igual probabilidad por cualquier nodo, podemos obtener la relación:

$$P^{t+1} = MP^t$$

donde M es la matriz de adyacencia normalizada del grafo de forma que sus columnas suman 1 (nótese que el elemento m_{ij} de esta matriz indica la probabilidad de pasar del nodo i al nodo j), $P^t = (p_1^t, \ldots, p_n^t)$ y $P^0 = (\frac{1}{n}, \ldots, \frac{1}{n})$.

En este caso, se puede probar que bajo condiciones relativamente comunes el vector P^t converge a un vector que verifica:

$$MP = \lambda P$$

que se corresponde con hallar el vector propio de la matriz de adyacencia normalizada M.

El algoritmo PageRank [177], utilizado para medir la relevancia de páginas web en Internet, es una variante de esta medida en la que, para eliminar puntos muertos en la evolución del caminante, se añade la probabilidad de que éste salte a una página aleatoria de la red en cualquier instante, a esta probabilidad la denotaremos por $1 - d$ (a d se le llama factor de amortiguación, *damping factor*).

Definición 19. *Fijado $d \in (0, 1)$, definimos la* centralidad Page-Rank *uni-relacional de un nodo, $u \in V$, como:*

$$PR(u) = \frac{1-d}{|V|} + d \sum_{(v,u)\in E} \frac{PR(v)}{gr_s(v)}$$

donde $gr_s(u)$ es el grado saliente de u.

Por supuesto, una primera opción a la hora de extender esta medida a grafos generalizados consiste en interpretar la red como un grafo uni-relacional (todas las aristas se consideran iguales) y calcular la centralidad como en el caso clásico, y de hecho es así como suele hacerse en la mayoría de los paquetes de cálculo de redes. Sin embargo, teniendo en cuenta las extensiones que hemos realizado anteriormente para los grados, es fácil dar ahora una extensión de esta definición que considere restricciones respecto de propiedades de los vecinos considerados:

Definición 20. *Fijado $d \in (0,1)$, $\alpha \in R$, y $s \in S$, definimos la* centralidad Page-Rank respecto de $\alpha = s$ *de un nodo, $u \in V$ como:*

$$PR_{\alpha=s}(u) = \frac{1-d}{|V|} + d \sum_{u\in\mathcal{N}^*(v)} \frac{PR(v)}{gr^V_{\alpha=s}(v)}$$

Y de forma similar a como hicimos con los grados, podemos considerar un Page-Rank medio si consideramos una medida de probabilidad asociada a los diversos valores que pueda tomar una función θ definida a partir de las propiedades de μ.

2.2.2. Medidas de Agrupamiento

Una de las medidas de agrupamiento más conocidas es el *Coeficiente de Clustering* [111](que también se conoce como *Coeficiente de Agrupamiento*) que, sobre un grafo unirelacional (se puede considerar dirigido o no), se define como sigue:

Definición 21. *Dado $u \in V$, el* Coeficiente de Clustering *de u se define como:*

$$C(u) = \frac{2 \cdot \#\{(v,w) \in E : \{v,w\} \subseteq \mathcal{N}(u)\}}{gr(u)(gr(u)-1)}$$

Que cuenta la proporción de triángulos (ciclos de longitud 3) con un vértice en u que hay en el grafo: de entre todos los posibles triángulos que se pueden formar usando u y los nodos de su entorno, aquellos que realmente están completos.

La posibilidades a la hora de extender la definición del coeficiente de clustering a grafos multirelacionales son amplias. Podemos reescribir la definición anterior de la siguiente forma:

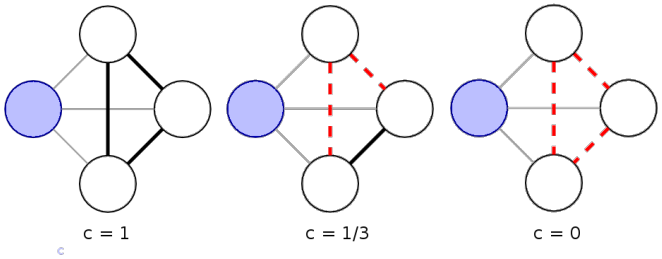


Figura 2.11: Coeficiente de Clustering.

$$C(v) = \frac{2 \cdot \#\{T(u,v,w) : \{v,w\} \subseteq \mathcal{N}(u)\}}{gr(v)(gr(v)-1)}$$

donde $T(u,v,w)$ es un triángulo formado por los nodos u, v y w.

Basta, pues, imponer condiciones sobre aristas y/o nodos de G a partir de las propiedades definidas en μ para obtener una definición del Coeficiente de Clustering que extiende a la anterior (que sería un caso particular) y que permite medir el agrupamiento que se produce en un grafo generalizado tomando en cuenta diversos aspectos del mismo.

$$C(v) = \frac{2 \cdot \#\{T_\theta(u,v,w) : \{v,w\} \subseteq \mathcal{N}_\theta(u)\}}{gr_\theta(v)(gr_\theta(v)-1)}$$

donde, en general, θ es un predicado que permite restringir el conjunto de elementos que se consideran en la muestra.

2.3. Aprendizaje Automático

El *Aprendizaje Automático* (*Machine Learning*, por su nombre en inglés, que es el término que se ha extendido más en los últimos años) es la rama de la Inteligencia Artificial que tiene como objetivo desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear algortimos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos [10, 216].

Aunque tiene ya algunos años de antigüedad y ha cosechado grandes éxitos en el pasado (y también algunos fracasos), ha sido recientemente cuando ha sufrido una explosión de interés por parte de la comunidad científica debido, principalmente, a los buenos resultados que ha obtenido en los últimos años. No

cabe duda que estos grandes resultados corresponden en gran medida a las intensivas labores de investigación, e inversión, de grandes empresas tecnológicas como Google, Apple, Microsoft, Facebook, Amazon y Baidu, pero sobre todo se debe al trabajo continuado de equipos de investigadores que, desde los organismos públicos, han realizado una investigación de carácter abierto y ajena a los requerimientos del mercado.

En muchas ocasiones, el campo de actuación del aprendizaje automático se solapa con el de *Data Mining* ya que las dos disciplinas están enfocadas en el análisis de datos. Sin embargo, el Aprendizaje Automático se centra más en el estudio de la complejidad computacional de los problemas con la intención de hacerlos factibles desde el punto de vista práctico, no únicamente teórico, y a la extracción de conocimiento a partir del algoritmo, mientras que el Data Mining se basa más en una extracción de información a partir de los datos, donde el objetivo último está en los datos, no en el algoritmo.

A un nivel muy básico, podríamos decir que una de las tareas del Aprendizaje Automático es intentar extraer conocimiento sobre algunas propiedades no observadas de un objeto basándose en las propiedades que sí han sido observadas de ese mismo objeto (o incluso de propiedades observadas en otros objetos similares)... o, en palabras más llanas, predecir comportamiento futuro a partir de lo que ha ocurrido en el pasado. Un ejemplo de mucha actualidad sería, por ejemplo, el de predecir si un determinado producto le va a gustar a un cliente basándonos en las valoraciones que ese mismo cliente ha hecho de otros productos que sí ha probado, y de la valoración de otros clientes que han tenido un comportamiento similar a él.

En cualquier caso, como el tema del que estamos hablando está relacionado con el aprendizaje, lo primero que hemos de preguntarnos es: *¿Qué entendemos por aprender?* y, ya que queremos dar metodologías generales para producir un aprendizaje de forma automática, una vez que fijemos este concepto habremos de dar métodos para medir el grado de éxito/fracaso de un aprendizaje. En cualquier caso, ya que estamos trasladando un concepto intuitivo, y que usamos normalmente en la vida diaria, a un contexto computacional, ha de tenerse en cuenta que todas las definiciones que demos de aprendizaje desde un punto de vista computacional, así como las diversas formas de medirlo, estarán íntimamente relacionadas con contextos muy concretos y posiblemente lejos de lo que, intuitivamente y de forma general, entendemos por aprendizaje.

Una definición relativamente general de aprendizaje dentro del contexto humano podría ser la siguiente:

> Proceso a través del cual se adquieren o modifican habilidades, destrezas, conocimientos, conductas o valores como resultado del estudio, la experiencia, la instrucción, el razonamiento y la observa-

ción.

De esta definición es importante hacer notar que el aprendizaje debe producirse a partir de la experiencia con el entorno. No se considera aprendizaje toda aquella habilidad o conocimiento que sean innatos en el individuo o que se adquieran como resultado del crecimiento natural de éste. Siguiendo un esquema similar, en el Aprendizaje Automático vamos a considerar *aprendizaje* a aquello que la máquina pueda aprender a partir de la experiencia, no a partir del reconocimiento de patrones programados a priori. Por tanto, una tarea central de cómo aplicar esta definición al contexto de la computación va a consistir en alimentar la experiencia de la máquina por medio de objetos con los que entrenarse (*ejemplos*) para, posteriormente, aplicar los patrones que haya reconocido sobre otros objetos distintos (en un sistema de recomendación de productos, un ejemplo sería un par particular cliente/producto, junto con la información acerca de la valoración que aquél haya hecho de éste).

En realidad, y desde un punto de vista práctico, en el Aprendizaje Automático confluyen muchas otras áreas de investigación, no todas estrictamente contenidas en la Inteligencia Artificial, y que aportan, de una u otra forma, un contenido teórico robusto a la vez que herramientas y problemas sobre los que poner en valor los avances realizados. Teniendo en cuenta que cualquier rama del conocimiento humano es susceptible de ser objeto de este tipo de técnicas, no debe resultar extraño que las influencias internas y externas de esta activa área de la computación sean cada vez más numerosas.

Hay un gran número de problemas que caen dentro de la disciplina del Aprendizaje Automático, y una de las diferencias que podemos destacar entre ellos estriba en el tipo de objetos que intenta predecir. En este sentido, una clasificación habitual es:

- *Problemas de Regresión*: Intentan predecir un valor real. Por ejemplo, predecir el valor de la bolsa a partir del comportamiento de la bolsa que está almacenado (pasado). O predecir el valor de una propiedad numérica de un nodo basándose en los valores de sus otras propiedades y las conexiones con el resto de elementos del grafo.
- *Problemas de Clasificación (binaria o multiclase)*: Intentan predecir la clasificación de objetos sobre un conjunto de clases prefijadas. Por ejemplo, clasificar si una determinada noticia es de deportes, entretenimiento, política, etc. o poder predecir el tipo de un nodo dentro de un grafo. Si solo se permiten 2 posibles clases, entonces se llama clasificación binaria; si se permiten más de 2 clases, estamos hablando de clasificación multiclase.
- *Problemas de Ranking*: Intentar predecir el orden óptimo de un conjunto de objetos según un orden de relevancia predefinido. Por ejemplo, el orden

en que un buscador devuelve recursos de internet como respuesta a una búsqueda de un usuario, o el orden que ocupan los nodos de un grafo respecto a la medida de Page-Rank que vimos en la sección anterior.

Normalmente, cuando se aborda un nuevo problema de Aprendizaje Automático, lo primero que se hace es enmarcarlo dentro de alguna de las clases anteriores, ya que la forma en que podemos medir el error cometido entre la predicción y la realidad dependerá de dicha clase. En consecuencia, el problema de medir cómo de acertado es el aprendizaje obtenido deberá ser tratado para cada caso particular de metodología aplicada, aunque en general podemos adelantar que necesitaremos "embeber"la representación del problema en un espacio en el que tengamos definida una medida.

Por otra parte, y dependiendo del tipo de patrón buscado y de cómo se aborde el tratamiento de los ejemplos, los diferentes algoritmos de aprendizaje se pueden agrupar en dos grandes bloques (en todo caso, no es extraño encontrar otros bloques adicionales que muestran ligeras diferencias con estos dos principales, como el semisupervisado, transducción, por refuerzo, etc.):

- *Aprendizaje supervisado*: se genera una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema, donde la base de conocimientos del sistema está formada por ejemplos etiquetados a priori (es decir, ejemplos de los que sabemos su clasificación correcta o su valor de regresión). Un ejemplo de este tipo de algoritmo es el problema de clasificación al que hemos hecho mención anteriormente.
- *Aprendizaje no supervisado*: donde el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formados únicamente por entradas al sistema, sin existir una clasificación/regresión correcta a priori (ni necesaria). Lo que se busca es que el sistema sea capaz de reconocer patrones que puedan determinar cómo se distribuyen los ejemplos y qué relaciones existen (en probabilidad) entre ellos.

Aunque puntualmente haremos uso de algoritmos de aprendizaje no supervisado, a lo largo de esta tesis nos centraremos casi siempre en problemas que requieren una aproximación supervisada y, por ello, la formalización que presentamos a continuación se ajusta principalmente a este tipo de aprendizaje supervisado.

2.3.1. Formalización del Aprendizaje

Como el concepto de aprendizaje automático es considerablemente amplio y puede ser enfocado desde diversos ángulos, nosotros elegiremos lo que hasta el

momento es la formalización más común y con mayores garantías de aplicación, y que se deriva de lo que se conoce como Aprendizaje Estadístico. Además, como hemos indicado, nos enfocaremos en dar aquí el marco para el aprendizaje supervisado.

Definición 22. *De forma general, el* problema de aprendizaje *al que nos enfrentamos es, dado un conjunto finito de ejemplos o muestras,* $S \subseteq X \times Y$ $(S = \{(x_i, y_i) : \ i \leq n\})$*, al que habitualmente se denomina* Conjunto de Entrenamiento*, y que es una muestra del proceso que se quiere aprender, encontrar* $h : X \to Y$ *tal que* $\forall (x_i, y_i) \in S \ (h(x_i) = y_i)$.

El problema específico del Aprendizaje Automático *es un caso particular del anterior, en el que restringimos* h *a ser una función computable por medio de un algoritmo,* $\mathcal{A}$*, de forma que dado un espacio de posibles funciones computables,* $\mathcal{H}$*, al que denominaremos* Familia de Hipótesis*,* $\mathcal{A}$ *es capaz de proporcionar, para cada posible conjunto de entrenamiento* S*, la* mejor *hipótesis,* $h_S \in \mathcal{H}$*, que resuelve el problema de aprendizaje:*

$$\begin{aligned} \mathcal{A} : \mathcal{P}_f(X \times Y) &\to \mathcal{H} \\ S &\mapsto h_S \end{aligned}$$

donde, para cualquier conjunto A*,* $\mathcal{P}_f(A)$ *representa las partes finitas de* A *(la colección de subconjuntos finitos de* A*).*

En realidad, dependiendo de las condiciones que impongamos sobre $\mathcal{A}$ y $\mathcal{H}$ obtendremos aproximaciones desde diversas disciplinas que trabajan sobre la extracción de patrones a partir de muestras parciales de un evento, y que pueden ir desde la Física, hasta el Aprendizaje Automático, pasando por el Análisis Funcional o el Aprendizaje Estadístico. En el Aprendizaje Automático, el foco está en el algoritmo $\mathcal{A}$, que muchas veces está asociado a la implementación en un lenguaje de programación específico y que genera un elemento concreto de la familia de hipótesis.

Además, y esta es la razón por la que decimos que la formalización presentada toma su base del Aprendizaje Estadístico, haremos una suposición adicional, y es que supondremos que existe una probabilidad, $\mathbb{P}$, definida en $X \times Y$, de forma que, aunque sea desconocida durante todo nuestro proceso, podemos ver las muestras de S como el resultado de n variables aleatorias independientes e idénticamente distribuidas sobre el espacio de datos (o, si queremos, como la ejecución de n veces la misma variable aleatoria). Esta suposición únicamente formaliza nuestra visión de que el proceso que estamos queriendo aprender sigue alguna ley desconocida pero existente.

Nos queda todavía por definir qué entendemos por la *mejor* hipótesis pero,

en cualquier caso, hemos de indicar que en la mayoría de las situaciones esta h_S ideal no se puede conseguir, bien sea por problemas en S (que podemos denominar como *ruido*), por limitaciones en $\mathcal{H}$ (que podemos denominar *sesgo*, ya que depende de nuestra presunción de que el problema se pueda modelar bajo ciertas circunstancias), o por problemas en el propio $\mathcal{A}$ (que podemos denominar *varianza*, ya que depende de la dificultad que tenga el algoritmo de encontrar el mejor h_S de entre todos los elementos disponibles). Es por ello que en la mayoría de los casos nos conformaremos con una solución que sea *suficientemente buena*.

Una forma de medir lo buena que es una solución es viendo el error que se comete entre el valor obtenido, $h_S(x_i)$, y el valor deseado, y_i, pero esta aproximación es en la práctica muy ineficiente ya que es relativamente fácil encontrar algoritmos que, bajo suposiciones mínimas, consigan reducir este error tanto como queramos en todo S.

En realidad, aunque en la práctica sea incalculable, lo que buscamos es minimizar el error que se comete con la *función real* que estamos intentando aprender. Para poder expresar esta medida, definimos lo que se conoce como error de predicción esperado:

Definición 23. *El* error de predicción esperado*, también conocido como* error de generalización *o* error de test*, de h_S es* [1]*:*

$$Err(h_S) = \mathbb{E}_{X,Y}[L(Y, h_S(X))]$$

donde L es una función de error que mide la discrepancia entre sus dos argumentos.

Este error mide cómo se comporta h_S sobre todos los posibles elementos de X, incluyendo no solo los del conjunto de entrenamiento, S, sino también aquellos no observados. Obviamente, el objetivo no es realizar las predicciones más acertadas sobre el conjunto de entrenamiento, sino aprender un modelo que sea capaz de comportarse correctamente en todo el espacio de posibles datos del problema.

Cuando tratamos un problema de clasificación la función de error más habitual es:

$$L(Y, h_S) = \begin{cases} 1 & \text{, si } Y = h_S(X) \\ 0 & \text{, si } Y \neq h_S(X) \end{cases}$$

[1] $\mathbb{E}_X[f(X)]$ denota el valor esperado de $f(x)$ (para $x \in \mathcal{X}$) con respecto a la distribución de probabilidad de la variable aleatoria X y se define como:

$$\mathbb{E}_X[f(X)] = \sum_{x \in X} P(X = x) f(x)$$

donde todas las clasificaciones erróneas son penalizadas de igual forma. En este caso, la generalización del error quedaría:

$$Err(h_S) = P(Y \neq h_S(X))$$

Cuando tratamos un problema de regresión la función de error más utilizada es la función de error cuadrático:

$$L(Y, h_S(X)) = (Y - h_S(X))^2$$

donde las diferencias más grandes entre el valor predicho y el valor correcto son penalizados de manera más intensa que las pequeñas. En este caso, el error de generalización quedaría:

$$Err(h_S) = \mathbb{E}_{X,Y}[(Y - h_S(X))^2]$$

El siguiente resultado nos dice que, incluso sin tener conocimientos a priori de la distribución $\mathbb{P}$, el error de cualquier hipótesis considerada puede descomponerse de forma informativa:

Teorema 1. *Para cada $h : X \to Y$ se tiene que:*

$$Err(h) = \mathbb{E}_X[(h(x) - r_{\mathbb{P}}(x))^2] + Err(r_{\mathbb{P}})$$

donde $r_{\mathbb{P}} = \mathbb{E}_Y[Y]$ se denomina función de regresión de $\mathbb{P}$.

En esta descomposición:

- El primer sumando, $\mathbb{E}_X[(h(x) - r_{\mathbb{P}}(x))^2]$, es el error que se comete al usar h como aproximación de la función de regresión.
- El segundo sumando, $Err(r_{\mathbb{P}})$, no depende de h, únicamente de $\mathbb{P}$, que es el modelo real que sigue el proceso que estamos intentando aprender, y refleja el ruido inherente en los datos, es decir, cómo de factible es que realmente esos datos sigan una distribución que pueda ser reflejada por medio de una relación funcional.

Por tanto, tomar $h = r_{\mathbb{P}}$ supone el error mínimo posible (es decir, la mejor solución), pero no podemos calcularla porque no conocemos $\mathbb{P}$.

2.3.2. Estimando el Error

Como ya hemos comentado, en la prática la distribución de probabilidad $\mathbb{P}$ es desconocida (si no fuese así, no sería necesario realizar un aprendizaje), por

lo que realizar la evaluación de $Err(h_S)$ es imposible. Además, normalmente es imposible evaluar datos adicionales a los presentes en S, ya que en la mayoría de los problemas, S constituye el único conjunto de datos disponible y sobre él el modelo debe realizar tanto el aprendizaje como la evaluación del error de predicción esperado. A continuación veremos una forma de estimar el error de generalización a pesar de esta cantidad de incógnitas en nuestro problema [229, **?**, 130, 162, 101, 21].

La pregunta que debe aparecer en todo proceso de aprendizaje con las condiciones expuestas es: ¿cómo puede $\mathcal{A}$ intentar minimizar el error en su búsqueda en el espacio de hipótesis, $\mathcal{H}$, si no sabe nada acerca de $\mathbb{P}$?

Para llevar a cabo esta tarea, disponemos de dos líneas de acción fundamentales:

1. El conocimiento a priori del problema (y, en consecuencia, de $\mathbb{P}$) puede reflejarse en la elección de $\mathcal{H}$, tanto en el conjunto de hipótesis que consideraremos como en el proceso de búsqueda que llevaremos a cabo para seleccionar la hipótesis más adecuada. Por ejemplo, y ha sido muy normal en estadística, cuando suponemos que podemos usar un modelo lineal para realizar el aprendizaje.

2. Aunque el error de generalización no se puede evaluar directamente, sí que podemos evaluar lo que se conoce como *error empírico* en S de las distintas hipótesis que podemos ir seleccionando:

$$R_S(h) = \frac{1}{n} \sum_{(x_i,y_i)\in S} L(y_i, h(x_i))$$

   En general, esta estimación del error es muy pobre debido a que la mayoría de algoritmos de aprendizaje automático van a conseguir minimizarlo (haciendo incluso que $R_S(h) = 0$). Pero intuitivamente, cuando n crece mucho, se espera que $R_S(h) \sim Err(h)$, por lo que minimizar R_S cuando el tamaño de S se va ampliando debe ser parecido a minimizar el error.

Normalmente, nos enfrentamos a un equilibrio entre el error de generalización y el de aproximación empírica: aunque una vía para minimizar el error de generalización puede ser enriquecer $\mathcal{A}$, los datos necesarios para mantener el error de aproximación bajo control crece rápidamente con el tamaño de $\mathcal{A}$.

Un problema similar lo muestra el siguiente teorema de descomposición del error, que se suele conocer como *Descomposición ruido/bias/varianza*:

Teorema 2. *Tomando S como una variable aleatoria que se distribuye siguiendo $\mathbb{P}^n$ (n aplicaciones de la distribución dada por $\mathbb{P}$), y definiendo $\overline{h}(x) =$*

$\mathbb{E}_S[h_S(x)]$ *como la predicción esperada para cada* $x \in X$ *fijo. Si* $\mathbb{E}_S[Err(h_S)] < \infty$*, entonces:*

$$\begin{aligned}\mathbb{E}_S[Err(h_S)] = \mathbb{E}_X[(Y - r_{\mathbb{P}}(x))^2] + \\ \mathbb{E}_{X\times Y}[(\overline{h}(X) - r_{\mathbb{P}}(X))^2] + \\ \mathbb{E}_{X\times Y}[\mathbb{E}_S[(h_S(X) - \overline{h}(X))^2]]\end{aligned}$$

En la descomposición anterior:

- $\mathbb{E}_X[(Y - r_{\mathbb{P}}(x))^2]$ solo depende de $\mathbb{P}$, y refleja el *ruido* inherente a los datos (similar a la obtenida en la descomposición anterior).
- $\mathbb{E}_{X\times Y}[(\overline{h}(X) - r_{\mathbb{P}}(X))^2]$ se denomina *bias o sesgo*, y refleja lo bien que se puede comportar la familia de hipótesis considerada, $\mathcal{H}$, para aproximar la mejor aproximación posible (el regresor).
- $\mathbb{E}_{X\times Y}[\mathbb{E}_S[(h_S(X) - \overline{h}(X))^2]]$ se denomina *varianza*, y refleja lo difícil que es conseguir el mejor aproximador de nuestra familia.

Igual que ocurrió con la descomposición anterior, ahora nos enfrentamos a un equilibrio entre dos de los sumandos: si enriquecemos $\mathcal{H}$ para conseguir que el sesgo disminuya, haremos más difícil encontrar la mejor hipótesis dentro de la familia, por lo que la varianza aumentará.

Hay otro importante concepto relacionado con las posibles modificaciones de estos sumandos, que se denomina *sobreajuste*, que se produce cuando una solución se ajusta tanto a S que hace que el error empírico sea excesivamente optimista, no mostrando que fuera de S no tiene porqué comportarse tan bien:

Definición 24. *Diremos que* $h \in \mathcal{H}$ sobreajusta *en* S *si:*

$$\exists h' \in \mathcal{H}\ (R_S(h) < R_S(h') \wedge Err(h) > Err(h'))$$

Todo lo que hemos visto relacionado con estas descomposiciones nos llama a buscar una familia $\mathcal{H}$ equilibrada, que sea capaz de comportarse bien incluso en el panorama de desconocimiento que el propio problema de aprendizaje impone. Para ello, se han desarrollado varias técnicas:

- Desde el punto de vista práctico, y debido a que el error empírico de h_S sobre el conjunto S suele ser bastante optimista, una posible aproximación empírica se realiza a través de un *conjunto de prueba*, lo que consiste en dividir el conjunto de entrenamiento, S, en dos subconjuntos disjuntos, S_{train} y S_{test}, llamados, respectivamente, *conjunto de entrenamiento* y

conjunto de prueba, utilizar S_{train} para realizar el aprendizaje, S_{test} para evaluar el error del modelo.

Una práctica habitual suele ser seleccionar el 70 % de los ejemplos de S para constituir S_{train} y el 30 % restante para constituir S_{test}, aunque la experiencia propone reducir progresivamente el conjunto de prueba conforme el tamaño de S crece [39].

En cualquier caso, se debe tener cuidado al dividir el conjunto S de tal manera que los ejemplos que acaben en S_{train} sean *independientes* de los que acaban en S_{test} y que ambos provengan de la misma distribución. Esto se puede conseguir normalmente haciendo una selección aleatoria de los elementos en S.

A pesar de que el error empírico calculado en el conjunto de prueba proporciona una estimación del error de generalización mucho más *fiable* que el calculado sobre el conjunto de entrenamiento, tiene el inconveniente de que reduce el número de ejemplos sobre los que el modelo aprende. Si S es lo suficientemente grande esto puede no ser un problema pero, en caso contrario, esta división puede dar lugar a un conjunto de entrenamiento demasiado pequeño como para llevar a cabo un aprendizaje correcto y, por tanto, para que la estimación del error de generalización sea fiable.

- Cuando S es pequeño, el *estimador basado en la validación cruzada* (K-Fold Cross Validation, en inglés) es preferible frente a la estimación basada en el conjunto de prueba. Esta estimación consiste en dividir S de manera aleatoria en K subconjuntos disjuntos, $S_1, ..., S_K$, y estimar el error de generalización como el promedio del error de predicción sobre los diferentes conjuntos S_k de los modelos $h_{S \setminus S_k}$, aprendidos sobre el resto de los ejemplos de S que no se encuentran en S_k [130].

- Para prevenir que $\mathcal{A}$ seleccione hipótesis demasiado complejas de $\mathcal{H}$, pero no evitar que existan para cuando sea necesario, se pueden usar versiones modificadas del método de optimización del error experimental.

  Uno de los procedimientos es el llamado *minimización del riesgo estructural*, que consiste en considerar una sucesión de familias de hipótesis, $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \mathcal{H}_3 \subseteq \ldots$, de complejidad creciente, y añadir al error experimental un factor que penaliza la complejidad de la clase seleccionada.

  Una variante de esta idea es la llamada *regularización*, en la que se toma una sola familia, $\mathcal{H}$, y una función que mide la complejidad de la hipótesis seleccionada (por ejemplo, si las hipótesis son parametrizaciones que pueden expresarse de forma vectorial a partir de una determinada construcción puede tomarse una norma de los parámetros).

En cualquier caso, seleccionar una familia $\mathcal{H}$ adecuada es más un arte que una ciencia, y depende del problema concreto y de la experiencia.

Además, hay resultados que demuestran que no hay una elección mejor a priori, como el teorema que popularmente se conoce como *No Free-lunch* [245], que nos dice que no hay una elección a priori de $\mathcal{H}$ que sea mejor que las demás, esto es, dadas dos familias, $\mathcal{H}_1$ y $\mathcal{H}_2$, siempre podremos encontrar problemas con conjuntos de entrenamiento en el que $\mathcal{H}_1$ se comporta mejor que $\mathcal{H}_2$, y viceversa.

2.3.3. Algunos modelos de aprendizaje

En esta última sección del capítulo introduciremos algunos de los modelos de Aprendizaje Automático que aparecerán en los desarrollos de los capítulos posteriores. La lista de modelos presentados no pretende ser exhaustiva ni cubrir ninguna categorización a priori, salvo la de hacer que este documento sea autocontenido (en la medida de lo posible).

Debido a que los árboles de decisión (y, en particular, el algoritmo ID3) y las redes neuronales se desarrollan generosamente en los capítulos correspondientes para mantener la coherencia de las modificaciones que se proponen en esta tesis, de estos modelos solo daremos aquí una presentación muy superficial.

K vecinos más cercanos

El algoritmo de los k *vecinos más cercanos* (k-*NN*, de k Nearest Neighbour, por su nombre en inglés) [59] es un sistema de clasificación supervisado basado en criterios de vecindad. Recordemos que los sistemas de clasificación supervisados son aquellos en los que, a partir de un conjunto de ejemplos clasificados, intentamos asignar una clasificación a un segundo conjunto de ejemplos.

En particular, k-NN se basa en la idea de que los nuevos ejemplos serán clasificados usando la clase a la cual pertenezca la mayor cantidad de vecinos más cercanos a ellos del conjunto de entrenamiento.

Este algoritmo explora todo el conocimiento almacenado en el conjunto de entrenamiento para determinar cuál será la clase a la que pertenece una nueva muestra, pero únicamente tiene en cuenta los vecinos más próximos a ella, por lo que es lógico pensar que es posible que no se esté aprovechando de forma eficiente toda la información que se podría extraer del conjunto de entrenamiento.

En problemas prácticos donde se aplica esta regla de clasificación se suele tomar un número k de vecinos impar para evitar posibles empates (aunque esta decisión solo resuelve el problema en clasificaciones binarias). En otras ocasio-

nes, en caso de empate, se selecciona la clase que verifique que sus representantes tengan la menor distancia media al ejemplo que se está clasificando. En última instancia, si se produce un empate, siempre se puede decidir aleatoriamente entre las clases con mayor representación.

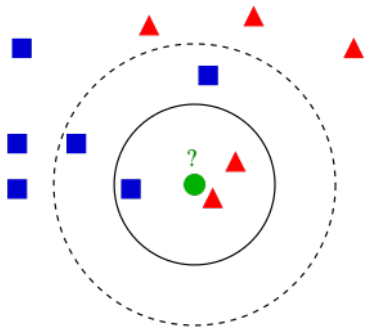


Figura 2.12: Funcionamiento del algoritmo k-NN.

Una posible variante de este algoritmo consiste en ponderar la contribución de cada vecino de acuerdo a la distancia entre él y el ejemplar a ser clasificado, dando mayor peso a los vecinos más cercanos.

Por ejemplo, podemos ponderar la clase de cada vecino de acuerdo al cuadrado inverso de sus distancias: Si x es el ejemplo que queremos clasificar, C es el conjunto de posibles clases, y $\{(x_i, y_i) : \ 1 \leq i \leq k\}$ es el conjunto de los k ejemplos de S más cercanos a x, definimos

$$w_i = \frac{1}{d(x, x_i)^2}$$

y, entonces, la clase asignada a x es aquella que verifique que la suma de los pesos de sus representantes sea máxima:

$$h_S(x) = \arg \max_{c \in C} \sum_{y_i = c} w_i$$

Esta mejora es muy efectiva en muchos problemas prácticos. Es robusto ante ruido en S y suficientemente efectivo cuando S es grande. Además, se puede ver que, al tomar promedios ponderados de los k vecinos más cercanos, el algoritmo puede evitar el impacto de ejemplos aislados con ruido.

Árboles de Decisión

Los árboles de decisión constituyen, quizás, uno de los modelos de clasificación más conocidos. Aunque se diseñaron en el marco de la Teoría de la Decisión

y la Estadística pronto se posicionaron como uno de los modelos más versátiles dentro del aprendizaje automático [203].

Originalmente fueron introducidos como modelos de clasificación, pero posteriormente se introdujeron variantes para resolver problemas de regresión. Dada la larga historia y el gran interés en los árboles de decisión, existen numerosas y buenas referencias que los estudian en profundidad [210, 161, 131].

Un árbol de decisión, que como su nombre indica, tiene estructura de árbol, está formado por un conjunto de nodos interiores, denominados *nodos de decisión*, y de nodos hoja, denominados nodos de respuesta y trabajan sobre datos que están compuestos por un conjunto de atributos y valores (normalmente en forma de tabla):

- Un nodo de decisión se asocia habitualmente a uno solo de los atributos y tiene tantas aristas salientes como posibles valores pueda tomar el atributo asociado. La interpretación semántica de un nodo de decisión es la de una pregunta que se le hace a la muestra analizada y, dependiendo de la respuesta que proporcione, para clasificar la muestra se pasa al nodo que se conecta por la arista correspondiente.
- Un nodo de respuesta está asociado a la clasificación que se quiere proporcionar, y devuelve la decisión del árbol con respecto al ejemplo de entrada.

Para ilustrar brevemente el concepto de árbol de decisión vamos a usar un ejemplo que clasifica un conjunto de datos sobre animales (presentado en la Tabla 2.1). Cada fila representa un animal, cada columna la existencia, o no, de alguna propiedad en ellos, y la última columna representa la clase a la que el animal pertenece ($Mamifero$, Pez o Ave), y que será la que queremos que el árbol de decisión aprenda a obtener a partir de las propiedades de los objetos. La figura 2.13 muestra un árbol de decisión que clasifica correctamente estas muestras.

Algunas de las ventajas que presentan los árboles de decisión como métodos de clasificación es que son fáciles de entender e interpretar, requieren de poco preprocesamiento de los datos para su construcción automática, son capaces de manejar datos de diversos tipos (tanto numéricos como categóricos), proporcionan un modelo de caja blanca que explica porqué devuelven los resultados que devuelven, funcionan bien ante presencia de ruido en las muestras, y escalan bien ante la presencia de conjuntos de entrenamiento muy grandes. Representan también la base de la variante que presentaremos para trabajar en el contexto de grafos con propiedades.

Se han desarrollado numerosos algoritmos para construir árboles de decisión de manera automática a partir de un conjunto de entrenamiento [160, 121, **?**]

Tabla 2.1: Datos de animales

		Huevos	Patas	Agua	Plumas	Tipo
1	**Ballena**	x	x	✓	x	MAM
2	**Atún**	✓	x	✓	x	PEZ
3	**Pato**	✓	✓	✓	✓	AVE
4	**Cigüeña**	✓	✓	x	✓	AVE
5	**Perro**	x	✓	x	x	MAM
6	**Bacalao**	✓	x	✓	x	PEZ

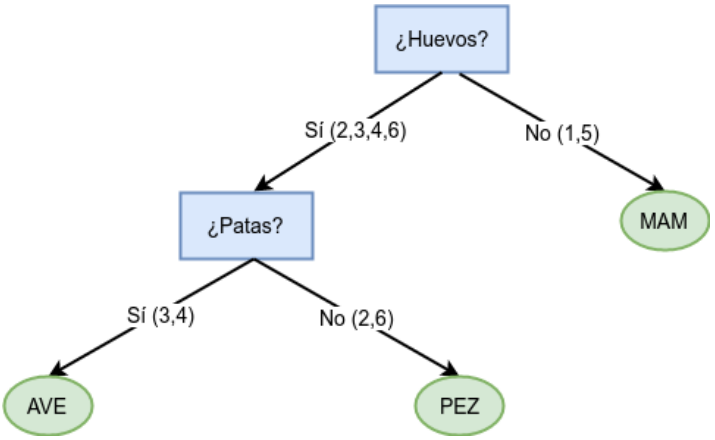


Figura 2.13: Árbol de decisión simple.

pero, sin duda, el más extendido y famoso es el algoritmo ID3, ya que constituye la base de la mayoría de algoritmos de este tipo desarrollados posteriormente.

Redes Neuronales Artificiales

Las *redes neuronales artificiales*, inspiradas en las estructuras neuronales de algunos seres vivos, se pueden expresar formalmente por medio de un grafo dirigido acíclico con propiedades $G = (V, E, \sigma, w, a)$, donde V es el conjunto de nodos (neuronas), E es el conjunto de aristas dirigidas, $w : E \to \mathbb{R}$ es la *función de pesos* asociados a las aristas del árbol, σ es una función de activación asociada a los nodos (normalmente, la misma para todos), y $a : V \to \mathbb{R}$ asocia a cada nodo el valor de activación del mismo.

Vamos a imponer una condición adicional, y es que V se pueda expresar como una unión disjunta de *capas*, $V_0, \ldots, V_k$, de forma que las conexiones solo

se dan entre capas sucesivas:

$$e = (u, v) \in E \text{ si y solo si } \exists i \ (u \in V_i \wedge v \in V_{i+1})$$

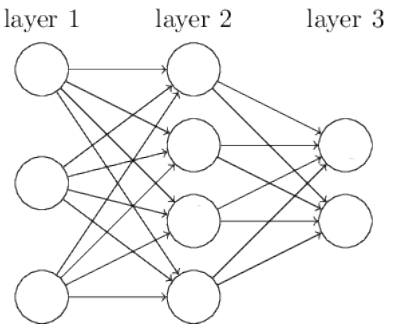


Figura 2.14: Red neuronal con una capa oculta.

Si $V_0 = \{i_1, \ldots, i_n\}$ (*capa de entrada*) y $V_k = \{o_1, \ldots, o_m\}$ (*capa de salida*), podemos asociar a cada red una función, $h_{(V,E,\sigma,w)} : \mathbb{R}^n \to \mathbb{R}^m$, de la siguiente forma:

1. Dados $(x_1, \ldots, x_n) \in \mathbb{R}^n$, definimos $a(i_1) = x_1, \ldots, a(i_n) = x_n$.
2. Calculado a para las capas $V_0, \ldots, V_{i-1}$, para cada $v \in V_i$ se define:

$$a(v) = \sigma_v \left(\sum_{e=(u,v)\in E} w(e)a(u) \right)$$

3. Entonces $h_{(V,E,\sigma,w)}(x_1, \ldots, x_n) = (a(o_1), \ldots, a(o_m))$.

A las capas V_i, con $0 < i < k$ se denominan *capas ocultas*. Al grafo (V, E, σ) se le denomina *estructura* de la red, porque es la parte estática de la red.

En estas condiciones, usar redes neuronales como modelo de aprendizaje significa usar como familia de hipótesis el conjunto de posibles redes que usan la misma estructura y que pueden variar en los pesos asociados a las aristas:

$$\mathcal{H}_{(V,E,\sigma)} = \{h_{(V,E,\sigma,w)} : \ w : E \to \mathbb{R}\}$$

Aunque no lo hemos explicitado, suele ser habitual usar una de las neuronas con un valor de activación fijo (normalmente, 1), denominada *nuerona de bias*, que está conectada con todas las demás neuronas y que sirve para introducir valores de umbral en el cálculo de la activación de las mismas.

Se han demostrado resultados muy interesantes acerca de la capacidad expresiva de estas redes para determinados objetivos, por ejemplo:

Teorema 3. *Para cada $n \in \mathbb{N}$, existe un grafo $(V, E, signo)$ que solo hace uso de 2 capas, tal que $\mathcal{H}_{(V,E,signo)}$ contiene todas las funciones de $\{-1,1\}^n$ en $\{-1,1\}$ (donde signo es la función signo habitual).*

Y, además, en estas condiciones, se puede probar que el número de neuronas, $|V|$, crece exponencialmente con n, el número de entradas.

Desde el punto de vista de capacidad computacional, el siguiente resultado establece algunas cotas interesantes:

Teorema 4. *Sea $T : \mathbb{N} \to \mathbb{N}$, y para cada n, sea $\mathcal{F}_n$ el conjunto de funciones que pueden ser calculadas por una máquina de Turing en tiempo acotado por $T(n)$. Entonces, existen constantes $b, c \in \mathbb{R}$ tales que para cada n existe una red (V_n, E_n) de tamaño acotado por $cT(n)^2 + b$ tal que $\mathcal{F}_n \subseteq \mathcal{H}_{(V,E,signo)}$.*

Los resultados anteriores hablan de la capacidad expresiva de las redes neuronales, pero no del método de entrenamiento, es decir, cómo calcular la red (los pesos) que proporciona la mejor aproximación.

El siguiente resultado muestra lo complicado que resulta encontrar estas redes incluso en los casos más simples:

Teorema 5. *Sea $k \geq 3$. Para cada n, sea $(V, E, signo)$ una red con 1 sola capa oculta, n neuronas de entrada, $k+1$ nodos en la capa oculta, y una solo neurona de salida. Entonces, encontrar la red de $\mathcal{H}_{(V,E,signo)}$ que minimiza el error empírico, R_S, es **NP**-duro.*

Debido a esta dificultad, hay que realizar búsquedas heurísticas para poder encontrar redes suficientemente buenas. A mediados de los años 80 se presentó un algoritmo, llamado *Retro-Propagación* (*Back Propagation*, en inglés), que aproxima en muchos casos los pesos a partir de S, y que se puede resumir muy brevemente en los siguientes puntos:

1. Empezar con w cualquiera (generalmente elegida al azar).
2. Seleccionar $(x, y) \in S_{train}$ al azar, y usar x como dato de entrada de la red.
3. Generar la salida, $o(x)$ usando x como entrada (propagación hacia delante, tal y como se explicó anteriormente).
4. Comparar la salida generada por al red con la salida deseada, $\Delta = |o(x) - y|$.
5. La diferencia obtenida entre la salida generada y la deseada se usa para ajustar los pesos, w, de las conexiones de las neuronas en la capa de salida.
6. El error se propaga hacia atrás, hacia la capa de neuronas anterior, y se usa para ajustar los pesos w de las conexiones entrantes en esta capa.

La forma de modificar estos pesos es por medio de algún procedimiento gradiente que busca minimizar el error.

7. Se continúa propagando el error hacia atrás y ajustando los pesos hasta que se alcance la capa de entrada.

8. Este proceso se repetirá con los diferentes elementos de S_{train} hasta que se verifique alguna condición adicional de parada (minimización del error global sobre S_{test}).

Debido a que la minimización se hace por medio de algún procedimiento basado en gradientes, este algoritmo es especialmente indicado cuando las funciones σ_v son diferenciables. Es por ello, y por la facilidad para calcular su derivada en un punto a partir del propio valor de la función en ese punto, que es tan común usar la función *sigmoide* como función de activación.

$$\sigma(x) = \frac{1}{1+e^{-x}}, \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

En general, cuando se usan otro tipo de redes u otras funciones de activación, se usa un método de optimización por gradiente, que se basa en la *diferenciación automática* (concretamente, en su versión inversa), y del que el método de propagación hacia atrás puede ser considerado un caso particular.

2.3.4. Métodos combinados de aprendizaje

En el campo del aprendizaje automático, los *métodos combinados* (o *métodos ensemble*) utilizan múltiples algoritmos de aprendizaje para obtener un rendimiento predictivo que mejore el que podría obtenerse por medio de cualquiera de los algoritmos de aprendizaje individuales que lo constituyen por separado [258].

Como ya comentamos, la tarea general de aprendizaje a partir de un conjunto limitado de datos es complicada y, ni siquiera teniendo la certeza de que en el espacio completo de hipótesis que estamos considerando exista una buena solución, podemos estar seguros de encontrarla.

La idea de los métodos combinados es considerar múltiples hipótesis simultáneamente para formar una hipótesis que, esperamos, se comporte mejor. El término de *métodos ensemble* se suele reservar para aquellas combinaciones que hacen uso de múltiples hipótesis pertenecientes a una misma familia, mientras que se usa el término más general de *sistemas de aprendizaje múltiple* cuando se permite que las hipótesis que se combinan provengan de diversas familias.

Una combinación de algoritmos de aprendizaje supervisado es en sí mismo un algoritmo de aprendizaje supervisado y puede ser entrenado y usado para hacer predicciones. Sin embargo, se debe tener en cuenta que una combinación de hipótesis de una determinada familia no es necesariamente una hipótesis de la misma familia, por lo que podríamos obtener mejores resultados que con los elementos individuales de la familia, aunque también podemos correr el riesgo de obtener un modelo sobreajustado si no se tienen algunas precauciones. En la práctica, la forma en que se seleccionan los modelos individuales que se combinan hacen uso de algunas técnicas que tienden a reducir los problemas relacionados con el exceso de ajuste de los datos de entrenamiento y mejoran la predicción conjunta.

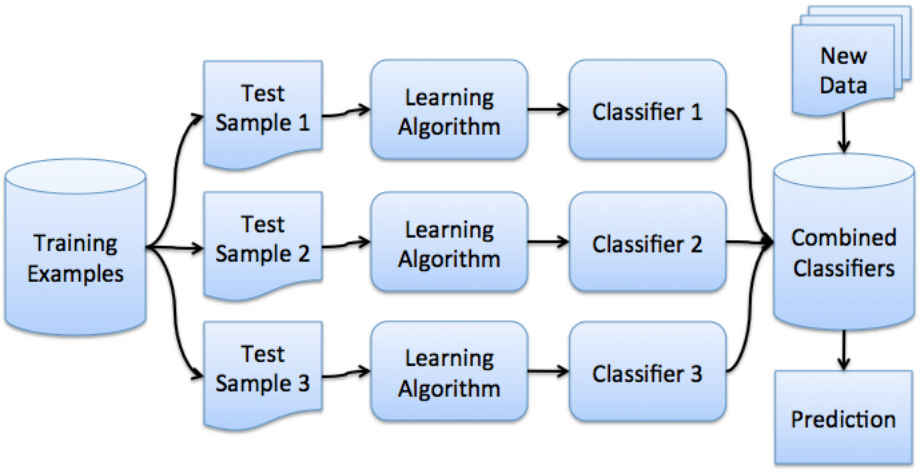


Figura 2.15: Métodos combinados de aprendizaje.

Empíricamente, se ha comprobado que cuando existe una diversidad significativa entre los modelos individuales, las combinaciones tienden a obtener mejores resultados, por lo que muchos de los métodos existentes buscan promover la diversidad entre los modelos que se combinan, y ello provoca a veces que se usen como modelos aquellos que hacen un uso fuerte de la aleatoriedad, en vez de modelos más dirigidos y que funcionan mejor individualmente.

Los métodos de combinación más usuales, y que comentaremos a continuación, son el *Bagging* (o *Agregación Bootstrap*), el *Boosting*, y los *Subespacios Aleatorios*.

Bagging

El Bagging es realmente un meta-algoritmo diseñado para conseguir combinaciones de modelos a partir de una familia inicial, provocando una disminución de la varianza y evitando el sobreajuste [42]. Aunque lo más común es aplicarlo con los métodos basados en árboles de decisión, se puede usar con cualquier familia.

La técnica consiste en lo siguiente:

- Dado un conjunto de entrenamiento, S, de tamaño n, el bagging genera m nuevos conjuntos de entrenamiento, S_i, de tamaño n', tomando al azar elementos de S de manera uniforme y con reemplazo, por tanto, algunos elementos del conjunto original pueden aparecer repetidos en los nuevos conjuntos generados.
- Si $n' = n$, entonces para valores de n suficientemente grandes, se espera que cada S_i tenga una fracción de $(1 - \frac{1}{e})$ ($\approx 63{,}2\,\%$) elementos únicos de S, y el resto son duplicados.
- A partir de estos m nuevos conjuntos de entrenamiento se construyen m nuevos modelos de aprendizaje.
- La respuesta final de la combinación se consigue por medio de la votación de las m respuestas (en caso de buscar una clasificación), o por la media de ellas (en caso de buscar una regresión).

Se ha probado que el bagging tiende a producir mejoras en los casos de modelos individuales inestables (como es el caso de las redes neuronales o los árboles de decisión), pero puede producir resultados mediocres o incluso empeorar los resultados con otros métodos, como es el caso de K-NN.

Boosting

A diferencia del bagging, en el boosting no se crean versiones del conjunto de entrenamiento, sino que se trabaja siempre con el conjunto completo de entrada y se manipulan los pesos de los datos para generar modelos distintos [213]. La idea es que en cada iteración se incremente el peso de los objetos mal clasificados por la hipótesis en esa iteración, por lo que en la construcción de la próxima hipótesis estos objetos serán más importantes y será más probable clasificarlos bien (aunque, quizás, en este proceso estamos empeorando el resultado en algún ejemplo que hasta ahora funcionaba bien).

El método de boosting más famoso es el que se conoce como AdaBoost que consta de los siguientes pasos:

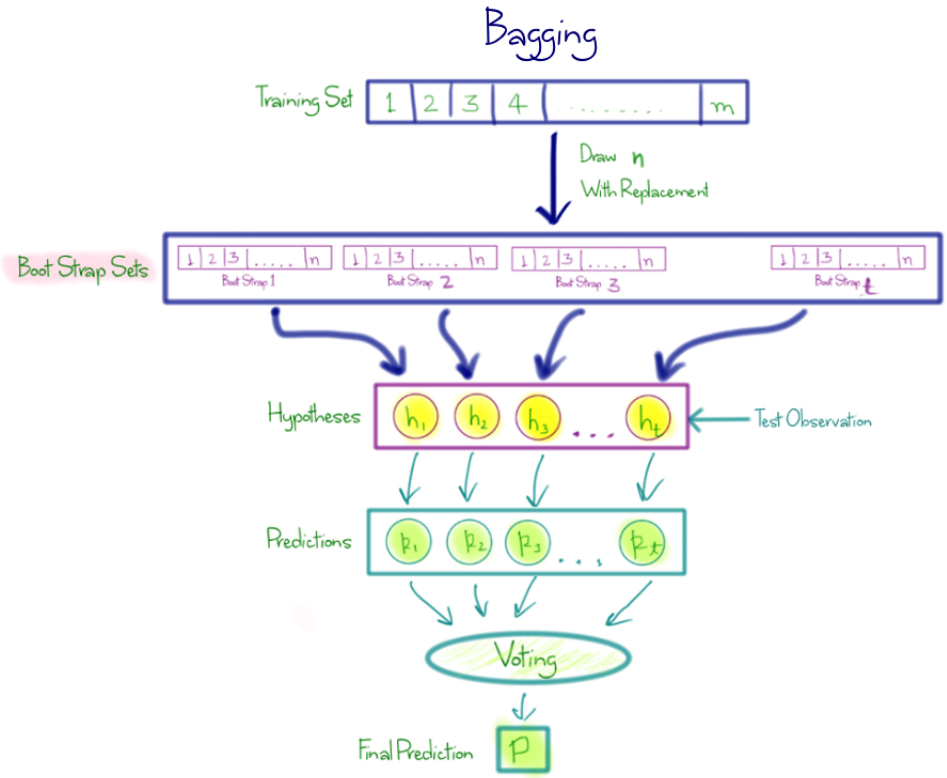


Figura 2.16: Representación gráfica del método Bagging.

1. Inicialmente a todos los datos del conjunto de entrenamiento se les asigna un peso idéntico, $w_i = \frac{1}{n}$, donde $n = |S|$. Se toma $t = 0$·
2. Se entrena el modelo usando S, y se obtiene $h_t = h_0$.
3. Se calcula $R_S(h_t)$, se cuentan cuántos objetos han sido mal clasificados y se identifica cuáles son, S_B.
4. Se incrementan los pesos en S_B.
5. Se entrena un nuevo modelo usando el conjunto de pesos modificados, h_{t+1}.
6. Si $t \leq T$, volver al punto 3.
7. El modelo final se consigue por votación ponderada usando los pesos de todos los modelos.

Concretamente, para modificar los pesos tras el cálculo de h_t con los pesos en el paso t, $w_{i,t}$, se utilizan las siguientes ecuaciones:

$$e_t = \frac{\sum_{i=1}^{n} w_{i,t} \cdot y_t \cdot h_t(x_i)}{\sum_{i=1}^{n} w_{i,t}}, \qquad \alpha_t = \frac{1}{2}\ln(\frac{1-e_t}{1+e_t})$$

donde x_i es el vector de entrada del objeto, e y_i es su clase/valor deseado.

Tras esto, los pesos son actualizados de la siguiente forma:

$$w_{i,t+1} = c \cdot w_{i,t} \cdot e^{-\alpha_t y_i h_t(x_i)}$$

donde c es una constante de normalización elegida de forma que $\sum_{i=1}^{n} w_{i,t+1} = 1$.

La combinación construida clasifica por medio del voto de la mayoría, ponderando cada modelo h_t por medio de α_t. Es decir:

$$h(x_i) = signo(\sum_{i=1}^{T} \alpha_t h_t(x_i))$$

Normalmente se espera una mejora significativa sobre la clasificación producida por cada uno de los modelos individuales, pero la convergencia no está garantizada y el rendimiento podría degradarse tras un cierto número de pasos.

Subespacios Aleatorios

En este método cada modelo se entrena con todos los ejemplos, pero solo considera un subconjunto de los atributos [110]. El tamaño de estos subconjuntos es el parámetro del método, y de nuevo el resultado es el promedio o votación de los resultados individuales de los modelos.

Más concretamente, si $X = X_1 \times \cdots \times X_D$, para cada sucesión $p = (i_1, \ldots, i_k) \in \mathbb{N}^k$ con $1 \leq i_1 < \cdots < i_k \leq D$ podemos considerar la proyección:

$$\begin{aligned} \pi_p : X &\to X_{i_1} \times \cdots \times X_{i_k} \\ (x_1, \ldots, x_D) &\mapsto (x_{i_1}, \ldots, x_{i_k}) \end{aligned}$$

Notaremos $S_p = \{(\pi_p(x), y) : \ (x, y) \in S\}$.

Si L es el número de hipótesis individuales que se desea obtener, la combinación se construye siguiendo el siguiente algoritmo:

1. Para cada $1 \leq i \leq L$:
   - Se selecciona al azar $1 \leq k \leq D$, y $p_i = (i_1, \ldots, i_k) \in \mathbb{N}^k$ con $1 \leq i_1 < \cdots < i_k \leq D$.
   - Se entrena h_i en S_{p_i} (siguiendo el algoritmo $\mathcal{A}_i$ elegido).

2. La combinación construida clasifica por medio del voto de la mayoría, es decir, para cada $x \in S$:

$$h(x) = signo(\sum_{i=1}^{L} h_i(x))$$

o devuelve la media de los valores dados por los hipótesis (también se puede dar una media ponderada):

$$h(x) = \frac{1}{L} \sum_{i=1}^{L} h_i(x)$$

Como resultado de este tipo de combinaciones, es especialmente interesante el modelo de aprendizaje Random Forest [141], un método ensemble a partir de árboles de decisión que hace uso de técnicas relacionadas con Bagging y Subespacios Aleatorios.

Capítulo 3

Consulta de patrones en grafos con propiedades

Debido al rápido crecimiento de las tecnologías relacionadas con Internet, y a que los grafos poseen una alta capacidad expresiva y son especialmente adecuados para modelar estructuras de elevada complejidad [9], el número de aplicaciones que hacen uso de ellos para modelar tanto sus datos como los procesos que los manipulan ha crecido espectacularmente en los últimos años.

Al igual que sucediera con los modelos de almacenamiento clásicos, el uso de nuevas estructuras conceptuales en las aplicaciones del *mundo real* requiere del desarrollo de nuevos sistemas para almacenar y consultar dichas estructuras, de tal forma que los usuarios puedan acceder a los datos almacenados de manera efectiva y eficiente. Como ocurre con toda nueva tecnología que se implementa, a pesar de que estas aproximaciones han demostrado una gran madurez pudiendo dar respuesta a muchas de las necesidades de los usuarios, todavía se encuentra en fase de desarrollo y en búsqueda de un conjunto de estándares que aseguren un crecimiento continuado y sin sobresaltos.

La creciente popularidad de las Bases de Datos en Grafo ha dado lugar a la aparición de interesantes problemas relacionados con el almacenamiento y consulta en este tipo de soluciones. Haciendo un paralelismo con la evolución tecnológica que vivieron los modelos de datos relacionales, estas bases de datos han asentado ya unos fundamentos robustos en lo referente a los accesos básicos de la información (creación, acceso, eliminación y modificación de elementos individuales de la estructura), pero todavía adolece de estándares robustos en algunas de las otras tareas necesarias para el almacenamiento y recuperación de la información, donde destacan principalmente las tareas relacionadas con mecanismos más avanzados de consulta.

Este capítulo se estructura como sigue. Tras dar unas notas generales acerca del problema de *detección de patrones*, comenzaremos haciendo un análisis del estado del arte de *Graph Pattern Matching*, recorriendo los conceptos fundamentales en este área y analizando las diferentes clasificaciones realizadas. A continuación haremos un repaso por las herramientas más importantes que permiten realizar consultas de patrones en grafos, que culminará con la presentación de nuestra propuesta, el *Property Query Graph* (PQG), presentando algunas propiedades generales de la herramienta propuesta y una colección de ejemplos que nos acercarán a posibles usos más elaborados.

3.1. Detección de patrones

Entre los problemas relacionados con el proceso de consulta en estas estructuras, la detección de patrones en grafos está considerada como uno de los problemas fundamentales, ya que engloba otros subproblemas necesarios para la obtención de sistemas de consulta potentes, como son la búsqueda de subgrafos, la búsqueda de caminos mínimos, o el estudio de la conectividad [259, 97].

Aunque a lo largo del capítulo daremos definiciones formales de los conceptos que necesitamos para abordar este problema desde un punto de vista computacional, comenzaremos dando una definición intuitiva de lo que se entiende por *patrón*.

Según la Real Academia Española de la Lengua, un *patrón es un modelo que sirve de muestra para sacar otra cosa igual*[1]. Esta definición exige una igualdad absoluta entre la estructura buscada y aquella que sirve de modelo original y, aunque puede ser válida en algunos contextos, limita su uso en el ámbito de las consultas de datos.

Por ello, estamos más interesados en trabajar con lo que en el contexto de Ciencia y Tecnología se conocen como *patrones estructurales*, que son *un tipo de sucesos u objetos recurrentes* y, en concreto, *aquella serie de propiedades identificables dentro de un conjunto mayor de datos*. Más generalmente, un *patrón abstracto es un modelo (o conjunto de reglas) que pueden ser usadas para generar o identificar entidades o partes de una entidad* [2]. En este sentido, la abstracción que proporciona se puede entender como la búsqueda, no de elementos concretos sino de aquellos que cumplan determinadas características que pueden ser satisfechas, bien por las propiedades que tienen, o bien por las relaciones en las que intervienen con los elementos que les rodean.

El proceso por el cual comprobamos la presencia de un determinado patrón

[1] `http://www.rae.es/`

[2] `https://es.wikipedia.org/wiki/Patron_abstracto`

en un conjunto concreto de datos se denomina *detección de patrones*, y computacionalmente comprende el conjunto de procesos mecánicos que permiten dar como respuesta una (o todas) las apariciones del patrón.

Cuando en un grafo concreto existe un patrón que se repite de manera recurrente, dicho patrón recibe el nombre de *motivo* (*network motif*, en ingleś). Muchos grafos asociados a sistemas complejos (que cuando se representan con grafos se conocen como *redes complejas*) suelen presentar motivos comunes, y se ha comprobado experimentalmente que los grafos obtenidos a partir de datos y problemas del mismo área de conocimiento suelen poseer también los mismos, o similares, motivos. Por ejemplo, los motivos compartidos por las redes de alimentos son distintos de los motivos compartidos por las redes genéticas o de los encontrados en la World Wide Web. De esta forma, los motivos pueden llegar a caracterizar clases universales de redes [156].

En los últimos años se han desarrollado numerosos algoritmos que permiten encontrar motivos en redes complejas, así como otros que permiten evaluar si un grafo contiene una subestructura determinada. En el año 2004 fue presentado un algoritmo de detección de motivos en redes que utiliza una técnica de conteo a través de fuerza bruta [157]. Este método funciona correctamente a la hora de encontrar motivos pequeños, pero tratar de encontrar motivos con más de 6 nodos no es computacionalmente posible. Kashtan et al. [120] presentaron también en el año 2004 el primer algoritmo de descubrimiento de motivos en redes basado en una técnica de muestro. Para ello, realizan un muestreo de las aristas a través de la red para estimar la concentración de una serie de subgrafos inducidos a lo largo del proceso de ejecución del algoritmo, consiguiendo detectar motivos a partir de un número muy pequeño de muestras en una gran variedad de redes. Además, la complejidad del algoritmo no depende del tamaño de la red, y consigue detectar motivos de mayor tamaño que el método presentado en [157]. En 2010, Ribeiro y Silva [194] presentan un nuevo algoritmo de descubrimiento de motivos en redes que hace uso de una estructura de datos en forma de árbol, denominada *g-trie*, que permite almacenar una colección de grafos (concretamente, cada camino en el árbol desde la raíz hasta una hoja representa un grafo concreto) y así reaprovecha los cálculos realizados previamente sobre estructuras pequeñas para cálculos de estructuras de mayor tamaño que las contienen, convirtiéndolo en el más eficiente en tiempo de los algoritmos de descubrimiento de motivos mencionados. Como contraparte, hace un uso excesivo de memoria, lo que impide que el algoritmo se ejecute de manera óptima en un ordenador personal. Otros algoritmos, como GBI [165], permiten el descubrimiento de motivos en grafos junto a la realización de otros objetivos complementarios.

Un caso particular de consulta, de menor envergadura que el reconocimiento de patrones general pero muy común, consiste en saber si existe un camino entre

un par de nodos de un grafo multi-relacional que cumpla con una determinada secuencia de tipos en las aristas que lo forman. Una forma de especificar dicha consulta es a través de una *expresión regular* definida sobre el alfabeto de tipos de las aristas [247]. Este tipo de consultas se enmarca en lo que se llama de forma general *Regular Pattern Matching*.

Teniendo en cuenta que nuestro marco de trabajo es más general que los grafos habituales (binarios y unirelacionales), queremos mencionar algunos algoritmos que han sido desarrollados para abordar el problema de detección de patrones en hipergrafos. En concreto, en [255] se trabaja con el problema de *probabilistic hypergraph matching* (detección de similitud entre dos hipergrafos basada en probabilidades) que, partiendo de una matriz de probabilidades de asociación entre pares de hiperaristas (una de cada hipergrafo que se está comparando), devuelve como resultado una matriz que indica la probabilidad de matching de cada vértice del primer hipergrafo con cada vértice del segundo. En [14] plantean el concepto de la *perfección* de un *matching* entre dos hipergrafos, midiéndola a partir del número de vértices de los hipergrafos que se pueden identificar de forma correcta. De forma más general, en [45] se analiza el problema de *hypergraph matching* extendiendo algunos de los algoritmos diseñados para grafos binarios al caso de los hipergrafos.

Cabe destacar que, a pesar de que las definiciones de grafo utilizadas durante esta tesis permiten trabajar de manera natural con hipergrafos, las definiciones que se presentan en este capítulo están orientadas a trabajar principalmente con grafos binarios, ya que es el modelo más habitual que se encuentra tanto en la mayoría de referencias como en los conjuntos de datos que servirán de validación de los algoritmos desarrollados. A pesar de esta restricción, muchas de las metodologías y definiciones que daremos se pueden ampliar con facilidad al caso más general, pero las comprobaciones experimentales serían más artificiales ya que el conjunto de ejemplos reales que hacen uso de hipergrafos es muy limitado.

3.2. Graph Pattern Matching

Como hemos visto en la sección anterior, la detección de patrones en grafos (*Graph Pattern Matching*) es un área de investigación activa desde hace más de 30 años que ha mostrado su utilidad en muy diversas áreas del conocimiento, desde la visión artificial, hasta la biología, pasando por la electrónica, el diseño asistido por ordenador, y el análisis de redes sociales, entre otros. Sin duda, su interés ha crecido aún más al hacer su aparición las bases de datos en grafo como una herramienta de estructuración y almacenamiento de la información de forma transversal común a todas las áreas de conocimiento. Por este motivo,

el problema de la detección de patrones en grafos se expande, con ligeras variantes, a través de diferentes comunidades científicas, mostrándose no como un problema único definido bajo una formalización común, sino como un conjunto de problemas relacionados.

Aunque, como hemos comentado, la definición de patrón puede variar entre comunidades, e incluso dentro de una misma comunidad de una línea de trabajo a otra, de forma general podemos enunciar el problema de detección de patrones en grafos como sigue, en sus variantes de cálculo o de decisión:

Definición 25. *Dado un patrón, Q, y un grafo, G, el* problema de cálculo asociado a la detección de patrones *para Q y G consiste en encontrar todas las ocurrencias de Q en G, denotadas como $M(Q, G)$, y que constituyen una familia de subgrafos de G.*

En las condiciones anteriores, el problema de decisión asociado a la detección de patrones *para Q y G consiste en comprobar si $M(Q, G) \neq \emptyset$.*

A veces, y dependiendo de las necesidades del problema que hace uso de él, podemos trabajar con una versión reducida que encuentra una ocurrencia de Q en G, $M^*(Q, G)$, y no todas las ocurrencias posibles. Obsérvese que, en este caso, M^* puede depender del algoritmo usado para resolverlo. Además, en el caso del problema de decisión, y por motivos de eficiencia computacional, suele hacerse la comprobación con M^*, y no con M. En el caso de la decisión será habitual interpretar Q como un predicado sobre grafos, y notaremos $Q(G)$ para indicar que el problema de decisión de detección del patrón Q en el grafo G tiene una respuesta afirmativa.

La definición exacta de qué se entiende por una *ocurrencia* de Q en G varía según cómo se defina el patrón. Cuando Q viene dado por una estructura de grafo (aunque no necesariamente usando los mismos conjuntos elementales de vértices y aristas que G) es habitual asociar esta ocurrencia con la existencia de identificaciones (quizás no tan fuertes como isomorfismos) entre Q y aquellos subgrafos de G que respeten algunas restricciones impuestas por Q [74].

Hagamos una rápida panorámica de diferentes aproximaciones a la detección de patrones en grafos, haciendo especial énfasis en aquellas que se relacionan con el problema de la detección de patrones *semánticos* en grafos con propiedades, por ser la tarea a la que nos enfrentamos en este trabajo.

Atendiendo a las restricciones que impone el patrón de búsqueda podemos realizar la siguiente clasificación:

1. *Detección estructural vs. Detección semántica* (*structural matching vs. semantic matching* [86]). Muchas soluciones al problema de detección de patrones en grafos están basadas estrictamente en la similitud estruc-

tural [232, 243]. Sin embargo, ya que los grafos pueden utilizarse como representaciones conceptuales, puede ser interesante encontrar patrones en grafos no sólo a partir de la similitud del grafo uni-relacional, sino también según la similitud de su interpretación en algún dominio de interés. Debido a que la carga semántica de un grafo con propiedades está almacenada en su mayoría en los tipos y los atributos de sus elementos, la detección estructural es a menudo insuficiente para encontrar similitudes conceptuales entre grafos. La detección semántica consiste en detectar subgrafos dentro de un grafo que cumplen tanto con la estructura como con los tipos y atributos presentes en los elementos del patrón [12, 54]. Haciendo uso de la notación que hemos presentado en nuestra definición de grafo, la detección estructural haría uso únicamente de la función γ, mientras que la detección semántica haría uso del resto de propiedades asociadas a nodos y aristas del grafo.

2. *Detección exacta vs. Detección inexacta.* Un algoritmo de detección de patrones puede devolver sólo los subgrafos de G que cumplen completamente con el patrón (detección exacta) o puede devolver una colección de subgrafos ordenada (*ranking*) por similitud con el patrón dado (detección inexacta) [86]. Los algoritmos de detección inexacta son denominados en ocasiones *correctores de error* ya que permiten detectar patrones en presencia de ruido [218, 228]. Además, algunos sistemas permiten detectar patrones sólo parcialmente especificados (utilizando símbolos especiales a modo de *comodines*, u operadores de cardinalidad) [31]. En este caso, a pesar de que los resultados cumplen completamente con el patrón dado, se considera una forma de detección inexacta ya que el patrón en sí es impreciso y no está unívocamente determinado.

3. *Solución óptima vs. Solución aproximada.* Los algoritmos de detección también pueden variar en términos de la calidad de la solución devuelta [86]. Los algoritmos óptimos garantizan encontrar una solución correcta (para la detección exacta, el conjunto de subgrafos que cumplen con el patrón; para la detección inexacta, el conjunto clasificado de los subgrafos que más se parecen al patrón), pero en la mayoría de los casos tienen un coste computacional muy elevado, llegando a ser intratables en el peor de los casos [219]. Los algoritmos de solución aproximada [52, 233, 227] a menudo presentan complejidad polinomial, pero no garantizan encontrar la(s) solución(es) correcta(s).

Junto a esta primera clasificación se puede dar otra atendiendo a la relación que debe cumplir el patrón con respecto a los subgrafos que se consideren ocurrencias del mismo. En todos los casos que se detallan a continuación se supone que Q puede ser identificado con una estructura de grafo subyacente,

(V_Q, E_Q, μ_Q), *de forma natural y unívoca* (donde, habitualmente, μ_Q contendrá predicados evaluables en los elementos de G). Además, supondremos que los subgrafos de G que se comprueban tienen la forma $G' = (V_{G'}, E_{G'}, \mu_{G'})$. Obsérvese que usamos la notación clásica para grafos (aristas como pares de nodos) ya que es la usada en los trabajos originales a los que hacemos referencia, pero puede ser fácilmente adaptada a la notación (y significado) más general que se ha introducido en este trabajo:

1. *Isomorfismo de Subgrafos* [232]. En este caso, $M(Q, G)$ está formado por todos los subgrafos de G que son isomorfos a Q. Es conocido que el isomorfismo de subgrafos es un problema NP-completo [243], por lo que esta opción suele ser muy costosa desde el punto de vista computacional.

2. *Graph Simulation* [155]. En este caso, $M(Q, G)$ está formado por todos los subgrafos $G' \subseteq G$ para los que existe una relación binaria $R \subseteq V_Q \times V_{G'}$ verificando:

   - Para cada nodo, $u \in V_Q$, existe al menos un nodo $v \in V_{G'}$ tal que $(u, v) \in R$ y $\tau(v) = \tau(u)$. Es decir, R identifica nodos del patrón con nodos del subgrafo manteniendo los tipos.
   - Por cada relación $(u, v) \in R$ y cada arista $(u, u') \in E_Q$, existe una arista $(v, v') \in G$ tal que $(u', v') \in R$. Es decir, R proyecta aristas del patrón en aristas del subgrafo.

3. *Bounded Simulation* [77, 259, 155]. Dada una función $f : E_Q \to \mathbb{N} \cup \{\infty\}$, $M(Q, G)$ está formado por todos los subgrafos $G' \subseteq G$ para los que existe una relación binaria $R \subseteq V_Q \times V_{G'}$ verificando:

   - Para todo nodo, $u \in V_Q$, existe al menos un nodo, $v \in V_{G'}$, tal que $(u, v) \in R$.
   - Para cada par $(u, v) \in R$:
     - v debe cumplir con los predicados impuestos por u.
     - Para cada arista $(u, u') \in E_Q$, existe $v' \in V_{G'}$ y $\rho \in \mathcal{P}_{G'}$, con $v \overset{\rho}{\rightsquigarrow} v'$ y tal que $(u', v') \in R$ y $|\rho| \leq f((u, u'))$.

4. *Regular Pattern Matching* [76, 193, 25, 155] (basado en Graph Simulation). Dada una expresión regular, $reg(e)$, para cada arista del patrón Q, $M(Q, G)$ está formado por todos los subgrafos $G' \subseteq G$ para los que existe una relación binaria $R \subseteq V_Q \times V_{G'}$ verificando:

   - Para cada nodo, $u \in V_Q$, existe un nodo $v \in V_{G'}$ tal que $(u, v) \in R$.

- Para cada par $(u, v) \in R$:
  - v cumple con los predicados impuestos por u.
  - Para cada arista, $e = (u, u') \in E_Q$, existe $v' \in V_{G'}$ y existe $\rho \in \mathcal{P}_{G'}$, con $sop_E(\rho) = (e_1, \ldots, e_n)$ tal que $(u', v') \in R$ y la secuencia $\tau(e_1) \ldots \tau(e_n)$ de tipos en las aristas es una cadena en el lenguaje de $reg(e)$.

5. *Edge Relationships* (caso particular de Regular Pattern Matching). Las aristas en los grafos con propiedades están normalmente *tipadas*, denotando el tipo de relación. En la práctica, usualmente se quieren encontrar patrones en grafos compuestos por aristas de varios tipos. Los tipos en las aristas pueden ser fácilmente incorporados a la detección de patrones extendiendo el concepto de Regular Pattern Matching con una forma restringida de expresiones regulares.

Algunas observaciones acerca de estas clasificaciones son [75]:

1. Debido a que el isomorfismo de grafos es un problema NP-completo [243], todos los algoritmos que se conocen para atacar este problema crecen exponencialmente en su complejidad con respecto al tamaño de los grafos de entrada. Por tanto, no es viable resolver el isomorfismo de grafos directamente en grafos grandes. Esto nos deja dos opciones para realizar detección de patrones en grafos por isomorfismos en un tiempo tratable: (1) utilizar un algoritmo de solución aproximada, o (2) utilizar un algoritmo de solución óptima pero aplicado sólo a un subconjunto de los posibles subgrafos en G [46]. Debido a que nuestro objetivo no será detectar cuándo un grafo G contiene un subgrafo que cumple con Q, sino dado un subgrafo en G verificar si cumple con Q, nos encontramos ante la segunda opción, por lo que sigue siendo viable utilizar algoritmos de solución óptima basados en isomorfismos.

2. En contraste con el Isomorfismo de Subgrafos, Bounded Simulation soporta (a) relaciones en lugar de funciones biyectivas entre los elementos, (b) predicados especificando condiciones de búsqueda basados en los contenidos de los nodos, y (c) proyectar aristas a caminos (acotados) en lugar de proyecciones arista a arista.

3. Graph Simulation puede verse como un caso restringido de Bounded Simulation en el que solo se permiten patrones simples en los que (a) los nodos poseen el tipo como único requisito presente en sus predicados asociados, y (b) todas las aristas tienen asociado el valor 1 (por lo que pasa a ser una proyección arista a arista).

4. En contraste con la intratable complejidad del Isomorfismo de Subgrafos, la detección de patrones en grafos basada en Bounded Simulation presenta complejidad cúbica. Concretamente, el problema de encontrar el patrón Q en G a través de Graph Simulation tiene complejidad $O((|V|+|V_Q|)(|E|+|E_Q|)$ (además, en la práctica Q suele ser pequeño en comparación con G, por lo que la complejidad termina dependiendo, esencialmente, de G) [75].

5. Se ha comprobado experimentalmente ([77, 76]) que la detección de patrones en grafos basada en Bounded Simulation permite identificar correctamente un gran número de comunidades en redes sociales reales que los otros métodos no han sido capaces de localizar. Por ello, se considera que, hasta el momento, es el método que permite mayor expresividad a la hora de hacer consultas en grafos.

6. Considerando las siguientes relaciones topológicas en la detección de patrones:

   a) Relaciones *hijo*. Si un nodo u en el patrón Q se asocia con el nodo v en el grafo G, entonces cada hijo de u en Q debe asociarse con un hijo de v en G.

   b) Relaciones *padre*. Si un nodo u en Q se asocia con un nodo v en el grafo G, entonces cada padre de u debe estar asociado con un padre de v en G.

   c) Conectividad. Si Q es conexo, entonces también lo son las ocurrencias de Q en G.

   d) Ciclos. Un ciclo no dirigido (respectivamente, dirigido) en Q debe asociarse con un ciclo no dirigido (respectivamente, dirigido) en G.

   Graph Simulation no preserva las relaciones de tipo padre, conectividad, ni ciclos no dirigidos, pero sí las relaciones de tipo hijo y los ciclos dirigidos. La detección basada en el Isomorfismo de Subgrafos preserva todas estas propiedades estructurales.

Debido a la complejidad existente en la detección de patrones en grafos se han desarrollado numerosas técnicas para tratar de reducirla. Algunas de ellas son: algoritmos incrementales, que permiten volver a evaluar la existencia de subgrafos que cumplen con un patrón Q en un grafo G sin necesidad de repetir todos los cálculos cuando G se actualiza [75]; técnicas que permiten buscar los subgrafos en un espacio más reducido (compresión de grafos) [75]; o técnicas de computación distribuidas, que permiten distribuir la carga de trabajo en varias máquinas o procesadores que trabajan en paralelo [75, 113].

3.3. Herramientas de consulta relacionadas

Los lenguajes de consulta han sido siempre la clave del éxito de los diferentes Sistemas de Gestión de Bases de Datos. El predominio que las Bases de Datos Relacionales han ejercido durante los últimos años se debe en gran parte al éxito de SQL (Structured Query Language), el lenguaje de consultas estructurado que se definió para ellas a finales de los años 70 [53, 231] y que facilitó la creación e implantación de sistemas de gestión que usaban el modelo relacional.

Esta correlación entre el éxito de un paradigma de almacenamiento y la existencia de un lenguaje de consulta adecuado ha hecho que, desde entonces, tras la aparición de cada nuevo sistema de gestión de base de datos, se haya creado un nuevo lenguaje de consulta para acceder a sus datos de forma adecuada. Este camino no siempre ha sido directo, y en numerosas ocasiones ha sido habitual la aparición de diversos lenguajes de consulta que luchaban por convertirse en el estándar. Por ejemplo, en el caso del formato RDF ha existido una seria competición entre múltiples lenguajes de consulta, como RQL, RDQL, SeRQL, hasta que finalmente parece haberse impuesto SPARQL como claro vencedor [112].

Los sistemas de almacenamiento basados en grafos no han sido ajenos a esta tendencia. Los lenguajes de consulta para bases de datos en grafo comenzaron a investigarse hace más de 20 años, cuando el uso de este paradigma de almacenamiento estaba restringido a experiencias académicas y puramente experimentales. Durante muchos años ha estado durmiendo el interés por crear un lenguaje de consulta adecuado, pero con la cantidad de datos disponibles en la actualidad que pueden sacar ventaja de este tipo de estructuras, donde destacan las redes sociales y la información que generan, ha habido un resurgimiento del interés por proporcionar un sistema de consulta adecuado y estandarizado [247].

Las herramientas actuales que permiten realizar consultas de patrones en grafos utilizan, o bien un lenguaje declarativo (como Cypher, SPARQL, o el propio SQL), o un lenguaje imperativo (con Gremlin como más claro representante). En el caso de los lenguajes declarativos, queda a responsabilidad del sistema (idealmente) el llevar a cabo la optimización automática de las consultas, preocupándose el usuario únicamente de declarar el objeto de su consulta. En el caso de las aproximaciones imperativas el plan de ejecución es responsabilidad del usuario (se espera que con conocimientos de desarrollo), por lo que suelen proporcionar aproximaciones a más bajo nivel.

A continuación presentaremos algunos de los lenguajes de consulta que soportan detección de patrones en grafos, analizando su capacidad expresiva, su potencia y sus características en cuanto a la capacidad de realizar este tipo de

consultas. La selección de las herramientas a analizar se ha hecho, o bien en base a la extensión de su uso, o bien por haber servido de fundamento a herramientas posteriores. En la Figura 3.1 se muestra P_{ex}, un patrón multi-relacional dirigido y etiquetado que será utilizado a modo de ejemplo para analizar algunas de las herramientas presentadas y comparar la capacidad expresiva para poder reconocerlo.

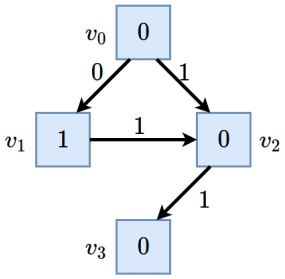


Figura 3.1: Patrón ejemplo P_{ex}.

3.3.1. XPath (XML Path Language)

Los orígenes de los lenguajes de marcado se remontan a la creación del Lenguaje de Marcado Generalizado (GML), desarrollado en la década de los 60 por Charles Goldfarb, Edward Mosher y Raymond Lorie para el formato de texto de IBM, SCRIPT [93].

Utilizando GML, un documento queda marcado a través de etiquetas que definen qué tipo de contenido se encuentra en cada región del documento, de esta forma puede ser formateado de manera automática para diferentes dispositivos. El marcado generalizado está basado en dos postulados:

- El marcado debe ser **declarativo**: debe describir la estructura del documento y otras propiedades, pero no especificar el procesado que debe hacerse sobre él.
- El marcado debe ser **riguroso**: de tal forma que las técnicas disponibles para procesar programas y bases de datos puedan ser utilizados para procesar documentos de este tipo también.

El lenguaje de marcado extensible (XML) es un lenguaje abierto de marcado que proviene de GML y que define una serie de reglas para codificar documentos en un formato que es legible, simultáneamente, por humanos y máquinas. Los

objetivos en el diseño de XML se idearon pensando en la simplicidad, la generalidad y la usabilidad en Internet [6]. Aunque el diseño de XML se centra en los documentos, el lenguaje es ampliamente usado para la representación de cualquier tipo de datos [80]. Se ha desarrollado una gran variedad de especificaciones de lenguajes para definir grafos usando XML, algunas de ellas son GraphXML [106], GraphML y XGMML [183]. A pesar de que XML define originalmente información estructurada en forma de árbol, los lenguajes para definir grafos sobre XML hacen uso de dicha estructura para almacenar la información de un grafo (a través de identificadores y referencias). Un documento XML puede ser analizado para saber si está bien formado (verificación sintáctica), o validado, para saber si cumple con un lenguaje específico definido en XML (verificación semántica).

XPath (XML Path Language) es un lenguaje de consulta declarativo definido por el World Wide Web Consortium (W3C) diseñado para seleccionar nodos de un documento XML. Además, XPath puede usarse para calcular valores (números, textos, valores lógicos, etc) desde un documento XML. El lenguaje XPath está basado en una representación en forma de árbol de un documento XML, y permite navegarlo seleccionando nodos bajo diversos criterios [8]. XPath utiliza *path expressions* para seleccionar los nodos en un documento XML, que son muy similares a las expresiones que se utilizan habitualmente para especificar la ruta de un fichero en un ordenador.

Para construir una consulta en Xpath se utilizan los siguientes elementos:

- *Eje*: permite seleccionar un subconjunto de nodos del documento, corresponde a recorridos en el árbol.
- *Predicado*: se escribe entre corchetes a continuación del eje. Si el eje ha seleccionado varios nodos, el predicado permite filtrarlos para dejar aquellos que cumplan determinadas condiciones.
- *Selección de campos*: se escribe a continuación del eje y del predicado. Si el eje y el predicado han seleccionado nodos, la selección de campos indica qué parte de esos nodos es la que extraemos.

En la Figura 3.2 se muestra un ejemplo de consulta XPath sobre un documento XML. En concreto, la consulta */wikimedia/projects/project/editions/*[2]* devolverá todos los objetos de tipo `editions` en la ruta especificada que ocupen la posición 2. Si hubiéramos añadido la cadena *@language* al final de la consulta obtendríamos el valor asociado al campo `language` de dichos objetos.

Como se puede observar en el ejemplo anterior, las consultas en XPath son relativamente sencillas de construir y de entender por un humano. XPath es un lenguaje de consulta para árboles XML que devuelve resultados exactos, por lo

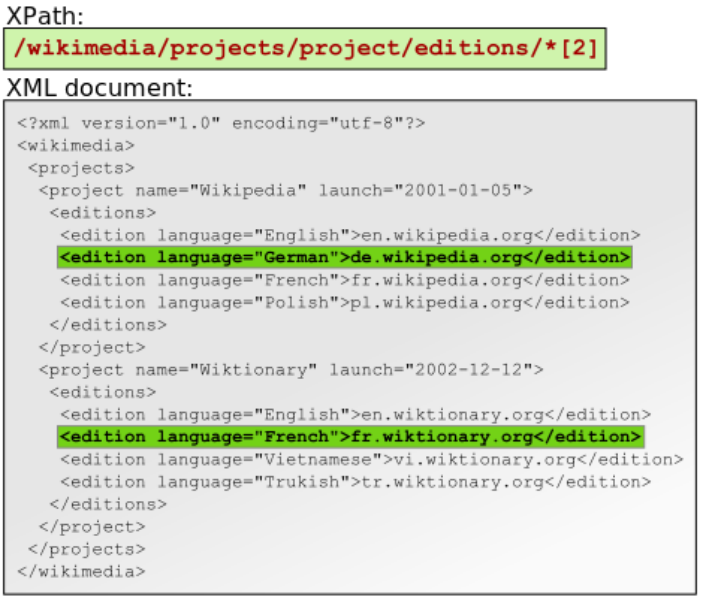


Figura 3.2: Consulta XPath.

que no está ideado para llevar a cabo consultas complejas de patrones en grafos. A pesar de que XPath puede referirse a un nodo a través de varios *padres*, no puede expresar un patrón con ciclos de manera directa. Sin embargo, gracias a las funciones anónimas que presentó la actualización XPath 3.0 se pueden escribir rutinas recursivas que detectan este tipo de patrones cíclicos. Además, se han ideado mecanismos que hacen uso de consultas XPath para realizar consultas de patrones en grafos cuando estos se encuentran almacenados en XML [47].

3.3.2. XQuery

XQuery es un lenguaje de consulta diseñado también para colecciones de datos XML, y proporciona los medios para extraer y manipular información en este tipo de documentos, o de cualquier fuente de datos que pueda ser representada mediante este formato. XQuery utiliza expresiones XPath para acceder a determinadas partes del documento XML, pero añade expresiones similares a las usadas en SQL (conocidas como expresiones FLWOR, ya que pueden estar compuestas por 5 tipos de sentencias: `FOR`, `LET`, `WHERE`, `ORDER BY` y `RETURN` [94]) y algunas capacidades de programación.

XQuery también incluye la posibilidad de construir nuevos documentos XML a partir de los resultados de la consulta. El siguiente ejemplo de código XQuery lista los personajes que aparecen en cada acto de la obra *Hamlet*, de Shakespeare,

obtenidas a partir del documento hamlet.xml [3]:

```
<html><head/><body>
{
  for $act in doc("hamlet.xml")//ACT
  let $speakers := distinct-values($act//SPEAKER)
  return
    <span>
      <h1>{ $act/TITLE/text() }</h1>
      <ul>
      {
        for $speaker in $speakers
        return <li>{ $speaker }</li>
      }
      </ul>
    </span>
}
</body></html>
```

Esta consulta XQuery, además de extraer los datos, construye un nuevo documento XML como respuesta. En este caso, el documento HTML construido tiene como raíz en su etiqueta `<body>` una etiqueta de tipo `<span>` de la que nacen los títulos de cada acto en etiquetas `<h1>` seguidos, cada uno de ellos, de una lista no ordenada (`<ul>`) con los personajes que intervienen en el mismo (cada uno de ellos inmerso en una etiqueta `<li>`). Las consultas XPath utilizadas en XQuery siguen siendo declarativas, pero las capacidades de programación que añade introducen un enfoque imperativo en este lenguaje.

Como puede observarse en la consulta ejemplo, XQuery mezcla la declaración de la consulta con el código que indica cómo debe representarse el resultado, por este motivo, las consultas en XQuery son menos intuitivas a la hora de ser analizadas a simple vista por un humano.

Debido a que XQuery utiliza *rutas* XPath para realizar sus consultas, no permite expresar directamente consultas generales de patrones en grafos. Sin embargo, se han llevado a cabo algunos desarrollos que han ampliado XQuery para que posea esta funcionalidad. En [196] presentan una implementación de Graph Pattern Matching estructural, exacto, óptimo y basado en el isomorfismo de subgrafos construida en Java sobre XQuery para minar bases de datos en RDF. Debido a que la consulta de patrones en grafos es habitualmente más

[3] http://www.ibiblio.org/xml/examples/shakespeare/hamlet.xml

costosa que la consulta de patrones jerárquicos en árboles, en [248] se propone un método que reduce el problema de consulta de patrones en grafos a consulta de patrones en árboles, consumiendo un grafo expresado en XML y transformando la consulta de patrón en grafo a una serie de consultas de patrones en el árbol subyacente XML. En [240], con el fin de permitir consultas sobre datos XML estructurados en forma de grafo de manera eficaz, se extiende XQuery agregándole características de representación de grafos y nuevas funcionalidades que permiten evaluar conceptos como conectividad y solapamiento, además se define un nuevo tipo de datos XGraph como una extensión de XPath para expresar consultas estructurales como un grafo general que puede contener ciclos.

3.3.3. SQL

SQL (Structured Query Language), siglas de Lenguaje de Consulta Estructurada, es un lenguaje declarativo para acceder a Sistemas de Gestión de Bases de Datos Relacionales (RDBMS). Una base de datos de este tipo se basa en un *modelo relacional* que se compone de varias *tablas*, y en el que cada tabla es a su vez un conjunto de *campos* (columnas) y *registros* (filas). Las relaciones entre tablas se representan por medio de claves almacenadas en campos.

SQL fue desarrollado inicialmente por IBM a principios de 1970 y no fue hasta finales de esa década cuando Relational Software (ahora Oracle) vio el potencial de los conceptos descritos en SQL y desarrolló su propio RDBMS basado en SQL con intención de comercializarlo. En junio de 1979, se presentó la primera implementación comercial de SQL.

El lenguaje SQL se compone de varios elementos, incluyendo:

- *Consultas/Sentencias*, que recuperan/modifican los datos de la base de datos según criterios específicos.
- *Cláusulas*, que son componentes constitutivos de las consultas y sentencias.
- *Expresiones*, que pueden producir valores escalares o tablas formadas por columnas y filas de datos.
- *Predicados*, que especifican condiciones, y se utilizan para limitar los efectos de las sentencias y las consultas o para cambiar el flujo de la ejecución.

En la Figura 3.3 se muestra una sentencia SQL a modo de ejemplo en el que se señalan varios de los elementos que la componen.

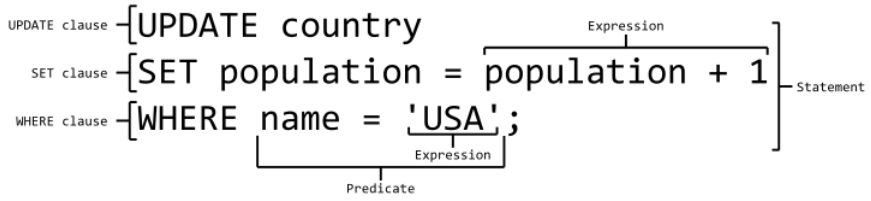


Figura 3.3: Consulta SQL.

La operación más común en SQL, la consulta, hace uso de la cláusula `SELECT`, utilizada para recuperar datos de una o más tablas en una base de datos relacional. Normalmente, la cláusula `SELECT` no tiene efectos persistentes en la base de datos. Al ser SQL un lenguaje declarativo, la consulta permite al usuario describir los datos que desea obtener de la base de datos, dejando que sea el RDBMS el que lleve a cabo la planificación, la optimización y la realización de las operaciones físicas necesarias para producir el resultado.

Una consulta incluye la lista de columnas que deben ser incluidas en el resultado final, normalmente después de la cláusula `SELECT`, con palabras claves y cláusulas opcionales que incluyen:

- La cláusula `FROM`, que indica la(s) tabla(s) de las que recuperar los datos. Puede incluir subcláusulas `JOIN` opcionales para especificar reglas para unir las tablas.
- La cláusula `WHERE`, que incluye un predicado de comparación que restringe las filas devueltas por la consulta.
- La cláusula `GROUP BY`, que proyecta filas que tienen valores comunes en un conjunto de filas más pequeño agrupando sus valores.
- La cláusula `ORDER BY`, que identifica qué columna utilizar para ordenar los datos resultantes y en qué dirección ordenarlos (ascendente o descendente).
- La palabra clave `DISTINCT`, que elimina los datos duplicados.

Una de las características más beneficiosas de los sistemas de bases de datos SQL, y la que supone uno de los mayores problemas en cuanto a la eficiencia de este tipo de sistemas, es la operación `JOIN`. Intuitivamente, la operación `JOIN` permite enlazar datos de dos o más tablas en el resultado de una única consulta. Las operaciones de tipo `JOIN` se encuentran en todas las consultas SQL

que tengan más de una tabla después de la palabra clave `FROM`. Por ejemplo, en la siguiente consulta:

```
SELECT customer_info.firstname, customer_info.lastname, purchases.item
FROM customer_info, purchases
WHERE customer_info.customer_number = purchases.customer_number;
```

se observa que ambas tablas (`customer_info` y `purchases`) deben tener una columna (un campo) denominada `customer_number` que contendrá el número de cliente único y se utilizará para unir las dos tablas en un único resultado. Intuitivamente, esta operación de tipo `JOIN` requiere que para cada registro de la tabla `customer_info` se recorra la tabla `purchases` al completo para encontrar los registros cuyo campo `customer_number` contiene el mismo valor que el campo `customer_number` del registro de la tabla `customer_info`. Cuando las tablas contienen muchos registros, este tipo de operaciones puede requerir demasiado tiempo de ejecución. Además, si la consulta requiere enlazar varias tablas (algo muy común en consultas de patrones en grafos), este hecho puede hacer que se produzca una explosión de complejidad y consumo de recursos de computación, lo que resulta en un aumento en los tiempos de respuesta de la consulta y puede provocar que su ejecución sea inviable desde un punto de vista práctico. Esencialmente, este problema se debe a que, a pesar de su nombre, las bases de datos relacionales no almacenan físicamente las relaciones entre los datos, haciéndolas inadecuadas para este tipo de tareas.

Los sistemas de gestión de bases de datos relacionales están dotados de planificadores y optimizadores cuyo objetivo es crear planes de ejecución de consultas óptimos. El planificador obtiene todas las posibles formas de *cruzar* (join) las relaciones que aparecen en una consulta, y es trabajo del optimizador estimar el coste de ejecutar cada una de ellas para encontrar la más económica.

Las bases de datos relacionales fueron diseñadas para datos tabulares con esquema fijo. Por ello, trabajan mejor en contextos que están bien definidos desde el principio y en el que las relaciones entre los registros son menos *importantes* que los registros en sí (las consultas se llevan a cabo habitualmente imponiendo restricciones sobre propiedades de los registros y no sobre las relaciones en las que participan). Sin embargo, intentar responder a preguntas en las que se involucran muchas relaciones entre los datos (por ejemplo, realizar consultas de patrones en grafos) con una base de datos relacional implica numerosas y costosas operaciones `JOIN` entre las tablas.

SQL es un lenguaje de consulta lo suficientemente potente como para poder expresar cualquier consulta de patrones en grafos con propiedades de los tipos presentados en este capítulo (aunque algunas de ellas requieren de funciones definidas por el usuario para poder ser llevadas a cabo).

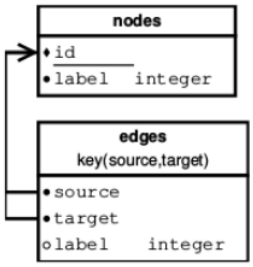


Figura 3.4: Esquema Base de Datos Relacional.

La Figura 3.4 muestra un modelo relacional que permite almacenar grafos dirigidos y etiquetados en un sistema relacional clásico. La clave primaria de la tabla *nodos* es el atributo *id*, esta tabla además contiene un campo para almacenar la *etiqueta*. La tabla de aristas posee dos atributos, *source* y *target*, que se corresponden con identificadores de nodos en la tabla correspondiente y que constituyen la clave primaria. La tabla de aristas también posee un campo para almacenar la *etiqueta*.

Un ejemplo de consulta SQL para encontrar todas las incidencias del patrón P_{ex} en un grafo almacenado a través de dicho modelo relacional podría ser:

```
SELECT v0.id, v1.id, v2.id, v3.id
FROM nodes v0, nodes v1, nodes v2, nodes v3, edges e0, edges e1, edges e2,
    edges e3
WHERE v0.label=0 AND v1.label=1 AND v2.label=0 AND v3.label = 0
AND v0.id<>v1.id AND v0.id<>v2.id AND v0.id<>v3.id
AND v1.id<>v2.id AND v1.id<>v3.id AND v2.id<>v3.id
AND e0.source=v0.id AND e0.target=v1.id AND e0.label=0
AND e1.source=v0.id AND e1.target=v2.id AND e1.label=1
AND e2.source=v1.id AND e2.target=v2.id AND e1.label=1
AND e3.source=v2.id AND e3.target=v3.id AND e1.label=1;
```

Para un patrón sencillo como P_{ex} esta consulta SQL requiere cuatro operaciones de tipo `JOIN` en la tabla de aristas con su respectivo coste en rendimiento. Además, a pesar de que se podría haber aplicado alguna optimización directa, como por ejemplo eliminar las restricciones que exigen que el identificador de v_1 sea diferente al de v_0 y v_2 (debido a que sus etiquetas son diferentes), este ejemplo muestra que SQL no es adecuado para este tipo de tareas, ya que genera consultas largas y poco intuitivas para un humano.

La forma más sencilla de generar consultas de este tipo en SQL es a través

de una sola sentencia. Desafortunadamente, el rendimiento de estas consultas es pobre ya que requiere una operación `JOIN` de la tabla de aristas para cada arista en el patrón. En [118] el rendimiento de consultas de patrones en grafos a través de instrucciones SQL individuales se contrasta con un enfoque que genera una instrucción SQL para cada camino en el patrón, utilizando un recorrido *depth first*. La consultas son traducidas a SQL recorriendo el patrón a consultar y construyendo varias instrucciones SQL, mejorando así el rendimiento de la consulta. Este tipo de problemas de rendimiento encontrados reflejan que tanto el planificador de consultas como el optimizador no están preparados para este escenario.

Conseguir que la ejecución de consultas de patrones en grafos a través de SQL sobre grafos almacenados en bases de datos relacionales tenga un rendimiento aceptable puede ser un problema difícil. Se ha demostrado que no se puede confiar en el optimizador SQL para ejecutar de manera eficiente sentencias SQL individuales que implementan consultas de patrones en grafos. También se ha demostrado que para obtener un rendimiento aceptable en la ejecución de consultas de patrones en grafos se debe generar código SQL que optimice la estructura de las consultas. Esta complejidad podría evitarse si el optimizador de consultas de las bases de datos relacionales se ampliara para manejar de manera eficiente consultas de este tipo [118].

Otro problema que aparece cuando se intenta llevar a cabo una consulta de patrones en grafos en SQL es la duplicación de resultados debido a la diferente *disposición* que pueden tomar los elementos devueltos por la consulta. Resultados que difieren sólo en su disposición geométrica no tienen interés y aumentan considerablemente el tamaño del resultado de la consulta. Para evitar la devolución de estos resultados duplicados, el código SQL que se genera para una consulta de patrones en grafos debe obligar a ordenar de alguna manera los elementos.

Las bases de datos relacionales han sido la opción de almacenamiento de la mayoría de los proyectos de software durante décadas y SQL su lenguaje de consulta por excelencia. Debido a que la mayoría de las colecciones de datos se han almacenado en bases de datos relacionales, los primeros sistemas de consulta de patrones en grafos con propiedades también usaron consultas SQL, como es el caso de los *Grafos de Selección* que veremos más adelante.

3.3.4. GraphLog

GraphLog es un lenguaje de consulta desarrollado en la década de los 90, basado en una representación en forma de grafo tanto de los datos como de las consultas [56] y que fue implementado sobre el lenguaje de programación lógico

Datalog [114]. Las consultas son representadas como grafos y las aristas en una consulta representan aristas o caminos en la base de datos, haciendo uso de expresiones regulares para dotarlos de mayor expresividad.

GraphLog trabaja con multi-grafos etiquetados (en vértices y aristas) y dirigidos. Un *Query Graph*, en este contexto, se define también como un multigrafo etiquetado y dirigido con una arista distinguida, sin nodos aislados y con las siguientes propiedades: (1) los nodos están etiquetados a través de variables; (2) cada arista está etiquetada con un predicado aplicado a una serie de variables y constantes, dichos predicados pueden estar seguidos del cierre positivo ($^+$); y (3) la arista distinguida solo puede estar asociada a un predicado no seguido del cierre positivo.

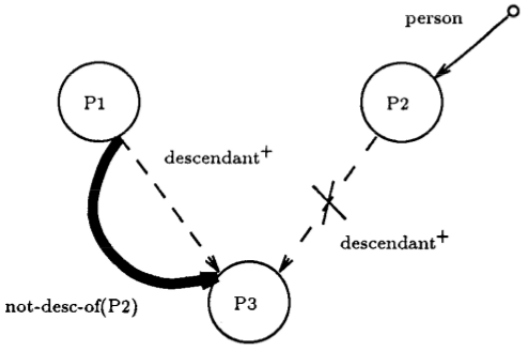


Figura 3.5: Ejemplo de Query Graph (GraphLog).

En la Figura 3.5 se muestra un Query Graph en el formato de GraphLog. La arista distinguida es representada mediante una línea más gruesa. El literal `descendant+`, que etiqueta la arista entre `P1` y `P3`, es un predicado seguido del cierre positivo (las aristas asociadas a un literal con el cierre positivo son representadas con líneas intermitentes). El literal `not descendant+`, etiquetando la arista entre `P2` y `P3`, es la negación de un literal con cierre positivo (representado a través de la arista tachada y dejando el literal en positivo). Por último, `person` es un predicado unario. Este query graph se corresponde con la consulta que devuelve un predicado ternario `not-desc-of(P1,P2,P3)` que se interpreta en lenguaje natural como: los descendientes, `P3`, de una persona, `P1`, que no son descendientes de otra persona, `P2`.

En general, la semántica que se asocia a un Query Graph en GraphLog se puede enunciar como sigue: si cada una de las aristas en el patrón se cumple, entonces la relación definida por la arista distinguida también debe ser cierta. Esta interpretación corresponde de manera natural a la forma en que leemos las

reglas de un programa lógico. Por ello, para traducir una consulta de este tipo a su sistema lógico subyacente, se debe asignar un predicado por arista.

En este sentido, GraphLog no es un sistema de consultas general en grafos, sino que está diseñado para inferir nuevas relaciones entre los datos (la indicada por la arista distinguida) a partir de las relaciones existentes. Dentro de la restricción comentada en cuanto al tipo de consultas, GraphLog representa un tipo de consulta de patrones en grafos con propiedades semántica, exacta, óptima y basada en el isomorfismo de subgrafos (a pesar de que permite un tipo de Regular Pattern Mathcing). Además, permite una representación gráfica de los patrones en grafos muy expresiva. Como contraparte, está limitado por el sistema lógico sobre el que se soporta y por la orientación de las consultas.

Podríamos intentar representar el patrón P_{ex} como un Query Graph en GraphLog de forma similar a como hemos dado su representación en grafo, donde las etiquetas de las aristas representarían un predicado involucrando sus dos nodos incidentes. Sin embargo, habría que producir una transformación adecuada que permitiera la transformación de los Query Graphs en GraphLog en herramientas de consulta más generales para poder llevar a cabo este tipo de consultas.

3.3.5. SPARQL

En el modelo de datos RDF (Resource Description Framework [4]) las entidades participan en tripletas (*sujeto*, *predicado*, *objeto*) que se interpretan como: *sujeto* está relacionado con *objeto* a través de la relación *predicado*. Los datos basados en RDF pueden ser interpretados como un grafo semántico en el que tanto los sujetos como los objetos representan nodos y los predicados representan relaciones entre ellos.

SPARQL es un lenguaje de consulta declarativo para atacar datos almacenados en formato RDF [215]. Fue convertido en estándar por el RDF Data Access Working Group (DAWG) del World Wide Web Consortium, y es reconocido como una de las tecnologías claves de la Web Semántica. Existen numerosas implementaciones para diferentes lenguajes de programación [5], así como herramientas que permiten construir consultas SPARQL de manera automática o que traducen consultas SPARQL a consultas en otros lenguajes [4, 3].

A continuación se presenta un ejemplo de consulta SPARQL que modela la pregunta "¿Cuáles son las capitales de los países de África?":

```
PREFIX ex: <http://example.com/exampleOntology#>
```

[4] https://www.w3.org/RDF/

```
SELECT ?capital
       ?pais
WHERE
  {
    ?x  ex:cityname     ?capital  ;
        ex:isCapitalOf  ?y        .
    ?y  ex:countryname  ?pais     ;
        ex:isInContinent ex:Africa .
  }
```

De forma similar a como funciona SQL, en SPARQL el bloque `WHERE` se utiliza para restringir la consulta. En la consulta anterior, `ex` identifica la ontología utilizada para interpretar `cityname`, `isCapitalOf` o `Africa`. `?capital` y `?pais` son las variables que almacenarán el nombre de cada capital y país. Por último, `?x` e `?y` son las variables que representan las entidades de tipo ciudad y país, respectivamente. La consulta anterior se interpreta como: selecciona valores para las variables `?capital` y `?pais` que verifiquen las cuatro tripletas descritas en el bloque `WHERE`. El procesador de consultas SPARQL se encarga de buscar los conjuntos de tripletas que cumplan con los cuatro patrones planteados, asignando los valores que cumplan con dichos patrones a las variables.

Para modelar el patrón P_{ex} en SPARQL podemos codificar el grafo en RDF modelando cada nodo como una entidad y las aristas como tripletas entre nodos conectados. El código SPARQL para realizar esta consulta podría ser:

```
SELECT ?V0 ?V1 ?V2 ?V3
WHERE
  {
    ?V0 e:0 ?V1 . ?V1 e:1 ?V2 . ?V2 e:1 ?V0 . ?V2 e:1 ?V3
    ?V0 rdfs:label 0 . ?V1 rdfs:label 1 . ?V2 rdfs:label 2 . ?V3 rdfs:label 0
    FILTER ( (?X0 != ?X1) && (?X0 != ?X2) &&
    (?X0 != ?X3) && (?X1 != ?X2) && (?X2 != ?X3))
  }
```

En esta consulta se utilizan dos ontologías, `e` y `rdfs`, que almacenan los tipos de las aristas y los conceptos `label` asociados a los nodos, respectivamente. Nótese que se ha añadido la cláusula `FILTER` con una serie de restricciones para asegurar la inyectividad en la proyección (isomorfismo de subgrafos). En el caso de que al patrón buscado no se le exija la inyectividad se podría prescindir de dicha cláusula.

Como puede observarse, la capacidad expresiva de una consulta de este tipo en SPARQL es muy superior a la que se obtiene con la consulta equivalente en SQL, pero queda lejos de ser intuitiva para un usuario humano.

SPARQL permite consultas de patrones estructurales, semánticos, óptimos, y exactos basados en el isomorfismo de subgrafos. A pesar de que SPARQL no permite realizar ningún tipo de Graph Simulation ni Regular Pattern Matching se han desarrollado extensiones del lenguaje como PSPARQL [13] que permiten consultar bases de datos RDF utilizando patrones basados en expresiones regulares. Además, su esquema de datos basado en grafo facilita la construcción de programas en otros lenguajes que generen código SPARQL para estos fines.

3.3.6. Gremlin

Gremlin es, simultáneamente, un lenguaje de consultas en grafo y una máquina virtual desarrollada por la Apache Software Foundation, especialmente orientado a Apache TinkerPop[5]. Gremlin está basado en Blueprints[6], lo que le permite trabajar con sistemas de persistencia que implementen dicha interfaz. Algunas bases de datos que implementan Blueprints son: Neo4j [7], OrientDB[8] y Titan [9]. Trabaja sobre bases de datos en grafo de tipo OLTP[10], así como para sistemas de análisis de grafos OLAP[11], y está construido sobre Groovy[12].

Como lenguaje, Gremlin soporta de manera natural consultas imperativas y declarativas (aunque su enfoque es imperativo) y es independiente a la base de datos utilizada, por lo que no se ajusta a un paradigma de programación concreto. Como máquina virtual, posee un planificador de consultas en tiempo de compilación y otro en tiempo de ejecución, un compilador/optimizador extensible, modelos de ejecución distribuidos, y es Turing completo [199].

Su sintaxis recuerda a la programación funcional, permitiendo realizar consultas declarativas e imperativas, ya que está basado en un flujo de datos que permite a los usuarios expresar de manera sencilla consultas complejas en gra-

[5]Apache TinkerPop es un framework de computación sobre grafos.

[6]Blueprints es una interfaz para el modelo de grafos con propiedades.

[7]`http://www.neo4j.com`

[8]`http://www.orientdb.com`

[9]`http://titan.thinkaurelius.com/`

[10]On-Line Transaction Processing, es un tipo de procesamiento que facilita y administra aplicaciones transaccionales, usualmente para entrada de datos, recuperación y procesamiento de transacciones.

[11]On-Line Analytical Processing, es una solución cuyo objetivo es agilizar la consulta de grandes cantidades de datos.

[12]Lenguaje de programación orientado a objetos implementado sobre la plataforma Java.

fos (no hipergrafos) con propiedades. Cada consulta en Gremlin está compuesta por una secuencia de pasos que realizan operaciones atómicas en el flujo de datos. Cada paso puede ser de tipo *map* (transformando los datos), de tipo *filter* (eliminando algunos de los objetos del flujo), o de tipo *side effect* (cálculo de valores auxiliares). Gremlin extiende estas tres operaciones fundamentales para proporcionar a los usuarios una rica colección de pasos que permiten definir cualquier pregunta concebible sobre los datos.

Una consulta en Gremlin es denominada *traversal*, y cada uno de los procesos de computación que se involucran en su ejecución son denominados *traversers*. Una consulta imperativa en Gremlin indica a los traversers cómo proceder en cada paso del traversal. Por ejemplo, el traversal imperativo que se muestra a continuación parte del conjunto de vértices del grafo y utiliza un paso para filtrar los vértices y obtener sólo aquellos cuyo nombre es `Juan`. El traverser se divide entonces entre todos los colaboradores de `Juan` que no son el propio `Juan` (han creado los mismos nodos). A continuación, los traversers seleccionan los administradores de esos colaboradores para finalmente contar cuántos administradores han colaborado con `Juan`. Esta consulta es imperativa ya que indica a los traversers qué hacer de manera explícita y procedural.

```
g.V().has("name","Juan").as("a").
  out("created").in("created").
    where(neq("a")).
  in("manages").
  groupCount().by("name")
```

Un traversal declarativo en Gremlin no indica a los traversers el orden en el que ejecutar su recorrido, sino que expresa el patrón que deben cumplir los diferentes elementos para satisfacer la consulta. Para producir este tipo de traversal declarativo en Gremlin se hace uso del paso `match()`, que es de tipo *map* y permite expresar de manera declarativa un patrón que se corresponde con el conjunto de nodos que debe cumplir. A través de este paso, el usuario puede describir un patrón que posee una serie de variables involucradas que deben cumplir con las restricciones impuestas. Cuando un traverser se encuentra en un paso de este tipo, un algoritmo propio de Gremlin analiza el estado actual del traverser (los objetos pertenecientes a él) y las estadísticas de los patrones, y devuelve un traversal que el traverser debe resolver. El algoritmo por defecto que lleva a cabo este proceso revisa de manera dinámica el plan de ejecución del patrón ordenando las diferentes restricciones impuestas de acuerdo a la capacidad de filtro (las restricciones que produzcan la mayor reducción del conjunto de nodos a tratar serán evaluadas primero).

En bases de datos grandes, donde el usuario no suele tener control sobre

las estadísticas del grafo, esta capacidad del método `match()` de ordenar las restricciones según su capacidad de filtro es una gran ventaja, ya que se obtiene una optimización de manera automática.

El traversal declarativo presentado a continuación produce el mismo resultado que el traversal imperativo anterior, pero con la ventaja de que no sólo aprovecha el planificador de consultas en tiempo de compilación (como los traversals imperativos), sino también el planificador de consultas en tiempo de ejecución que elige el paso óptimo para ser ejecutado en cada caso basado en las estadísticas históricas, favoreciendo aquellos pasos que tienden a reducir/filtrar la mayoría de los datos.

```
g.V().match(
  as("a").has("name","Juan"),
  as("a").out("created").as("b"),
  as("b").in("created").as("c"),
  as("c").in("manages").as("d"),
    where("a",neq("c"))).
  select("d").
  groupCount().by("name")
```

Cuando se compila el traversal, y dependiendo del motor de ejecución subyacente (una base de datos en grafo OLTP, o un procesador de grafos OLAP), la consulta del usuario es reescrita de tal manera que el plan de ejecución se optimiza basándose en un análisis de los costos de acceso a los datos del grafo y en las capacidades de los sistemas de datos subyacentes.

A continuación presentamos un ejemplo de consulta declarativa en Gremlin que solicita las incidencias del patrón P_{ex} en un grafo con propiedades (donde suponemos que las etiquetas se encuentran almacenadas en la propiedad `label` de los elementos):

```
g.V().match(
  as("v0").has("label","0"),
  as("v0").out("0").as("v1"),
  as("v0").in("1").as("v2"),
  as("v1").has("label","1"),
  as("v1").out("1").as("v2),
  as("v2").has("label","0"),
  as("v2").out("1").as("v3),
  as("v3").has("label","0"),
```

```
as("c").in("manages").as("d"),
where("v0",neq("v1")),
where("v0",neq("v2")),
where("v0",neq("v3")),
where("v1",neq("v2")),
where("v1",neq("v3")),
where("v2",neq("v3")))
```

Esta consulta solicita sólo los vértices, y no las relaciones que participan, pero con pocas modificaciones se podrían incluir también éstos en la respuesta de la consulta.

Como puede observarse, el método `match()` de Gremlin es muy útil para crear consultas de patrones en grafos con propiedades. Además, este método puede ser utilizado como un traverser más dentro de un flujo de instrucciones de tipo imperativo. También se observa que es un lenguaje expresivo, requiriendo poco código para expresar consultas complejas. Además, al ser Turing completo, Gremlin puede realizar cualquier tipo de consultas en grafos con propiedades, convirtiéndolo en un lenguaje muy adecuado para las consultas de patrones en grafos.

Gremlin permite realizar consultas semánticas, exactas, óptimas, y basadas en el Isomorfismo de Subgrafos de manera natural. Dado que Gremlin es un lenguaje, un juego de instrucciones y una máquina virtual, es posible diseñar otros lenguajes de consulta en grafos que compilen al lenguaje Gremlin. Por ejemplo, el lenguaje de consulta de patrones en grafos SPARQL puede ser compilado para ejecutarse en una máquina Gremlin[13].

3.3.7. Grafos de Selección

Knobbes et al. [129] desarrollaron un tipo de patrones multi-relacionales para consultar bases de datos basadas en la tecnología SQL. Las representaciones en forma de grafo de estos patrones son llamadas *grafos de selección*. Los grafos de selección representan los cimientos sobre los que está construido el algoritmo de inducción de árboles de decisión multi-relacionales MRDTL [139] (que será presentado con detalle en el siguiente capítulo) así como otros algoritmos de aprendizaje automático multi-relacionales [176].

Los principios que se persiguen con la definición de los grafos de selección, y que determinan su definición formal, son similares a los que seguiremos en

[13] `https://github.com/dkuppitz/sparql-gremlin`

nuestra definición posterior de consulta para grafos generalizados:

- **Reflejar la estructura del modelo**. Que la representación sea cercana a la estructura relacional permite entender mejor el patrón. Además, a nivel de implementación, es más fácil comprobar la cobertura de un objeto si la estructura de datos subyacente corresponde a la estructura del patrón.
- **Ser intuitivo**. Las descripciones lógicas estructurales o de primer orden, especialmente aquellas que involucran estructuras complejas y negaciones, son notoriamente difíciles de interpretar por humanos. En consecuencia, es deseable cualquier proceso que facilite la comprensión de estos patrones por los usuarios.
- **Soportar refinamientos locales y atómicos**. Debido a que es común realizar búsquedas por el espacio de posibles grafos de selección (los grafos de selección se utilizan en algoritmos que hacen búsquedas de este tipo), se necesitan operadores que permitan modificar a través de pequeños cambios un grafo de selección dado.
- **Permitir refinamientos complementarios**. Si la aplicación de un operador particular produce un cierto subconjunto de registros, también debería existir un operador complementario que produzca el subconjunto complementario (respecto del patrón original). Esta restricción deriva del hecho de que el objetivo final de los grafos de selección es ser utilizados en procedimientos de búsqueda de patrones tipo *top-down* [129].

A partir de estos requerimientos, un *Grafo de Selección* se define formalmente como un grafo dirigido donde:

- El conjunto de vértices viene dado por un conjunto de tripletas de la forma `(T,C,s)`, llamadas *nodos de selección*, donde `T` es una tabla del modelo relacional de datos, `C` es un conjunto (posiblemente vacío) de predicados simples sobre los atributos de `T` (si `A` es un atributo de la tabla `T`, se permiten predicados del tipo `A = valor`, `A < valor`, etc.), y `s` toma dos posibles valores: `abierto` o `cerrado`.
- El conjunto de aristas viene dado por un conjunto de tuplas de la forma `(p, q, a, e)`, llamadas *aristas de selección*, donde `p` y `q` son nodos de selección, `a` es una relación existente entre las tablas asociadas a `p` y `q` en el modelo de datos, y `e` toma dos posibles valores: `presente` o `ausente`.

Dado el esquema de datos relacional presentado en la Figura 3.6, la Figura 3.7 muestra un grafo de selección que selecciona a los padres que tienen un niño con un juguete, pero que no tienen un niño que sea varón, con un juguete.

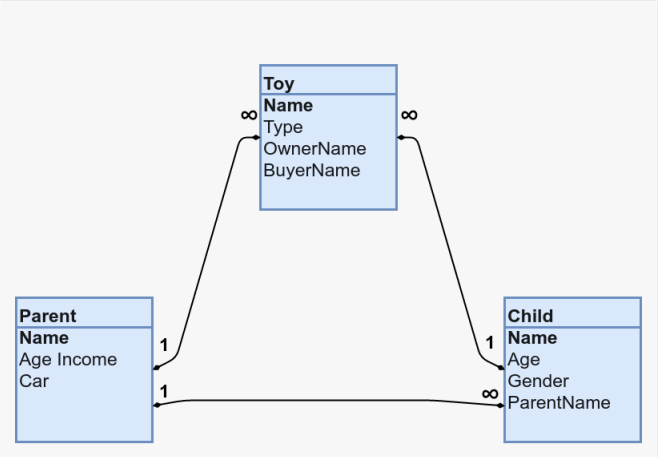


Figura 3.6: Esquema de datos relacional.

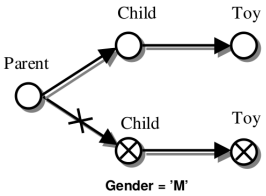


Figura 3.7: Grafo de selección.

Obsérvese que tachamos, o no, nodos (aristas) para indicar si son `cerrados` o `abiertos` (`presentes` o `ausentes`).

En general, la interpretación semántica de un grafo de selección es:

- Una arista `(p, q, a, presente)` que conecta el nodo `p` con un nodo `q` a tavés de una relación `a` combinada con una lista de condiciones en `q`, selecciona aquellos nodos en `p.T` que están conectados con al menos un nodo en `q.T` a través de una relación de tipo `a` que respeta la lista de condiciones `q.C`.
- Una arista `(p, q, a, ausente)` que conecta el nodo `p` con un nodo `q` a tavés de una relación `a` combinada con una lista de condiciones en `q`, selecciona aquellos nodos en `p.T` que no están conectados con un nodo en `q.T` a través de una relación de tipo `a` que respeta la lista de condiciones `q.C`.
- Cualquier subgrafo que es accesible por un nodo abierto a través de una arista ausente debe considerarse como un conjunto de condiciones negativas.

Siguiendo [129], el Algoritmo 3.8 (que a su vez hace uso del Algoritmo 3.9) muestra cómo un grafo de selección puede ser traducido a SQL. Esta traducción proporciona una semántica formal para los grafos de selección y proporciona, en última instancia, la formalización de cómo las consultas a través de grafos de selección deben ser llevadas a cabo en bases de datos relacionales:

El algoritmo produce una lista de tablas (`table_list`), una lista de condiciones *join* (`join_list`) y una lista de condiciones (`condition_list`) y las combina para producir una sentencia SQL. El algoritmo produce una operación de tipo *join* por cada tabla asociada a un nodo abierto. Para cada subgrafo accesible a través de una arista ausente, se produce una subconsulta llamando al procedimiento `translate_subgraph` que será utilizada para verificar si el nodo target está conectado con algún subgrafo con esas características. El hecho de utilizar `select distinct` en la consulta principal en lugar de `select` es debido a que un registro en `t_0` puede ser cubierto por el grafo de selección de múltiples maneras (diferentes disposiciones geométricas del patrón) pero debe ser contado sólo una vez (indicando que verifica el patrón).

Es habitual que las sentencias SQL resultantes no se utilicen directamente en los sistemas habituales de bases de datos relacionales, ya que una arquitectura específica puede llevar a cabo la consulta de manera mucho más eficiente. Por motivos de eficiencia, los grafos de selección tal y como se presentan en [129] no permiten la existencia de ciclos.

La aplicación del algoritmo `translate` sobre el grafo de selección dado en la Figura 3.7 produce la siguiente sentencia SQL:

```
select distinct T0.Name
from Parent T0, Child T1, Toy T2
where T0.Name = T1.ParentName and T1.Name = T2.OwnerName
and T0.Name not in
    (select T3.ParentName
    from Child T3, Toy T4
    where T3.Name = T4.OwnerName and T3.Gender = 'M')
```

Los grafos de selección representan un tipo de Graph Pattern Matching exacto, óptimo y basado en el isomorfismo semántico de grafos. Además, los grafos de selección permiten una representación gráfica muy expresiva y pueden ser construidos en pasos sucesivos a través de refinamientos (como veremos en el siguiente capítulo), ofreciendo buenas condiciones para ser utilizados en procedimientos de descubrimiento. Como contraparte, presentan el inconveniente de que los patrones que representan no pueden contener ciclos, limitando así la potencia expresiva de las consultas. Además, al ser una representación gráfica de

```
translate(S : Selection Graph)

    table_list := ''
    condition_list := ''
    join_list := ''
    for each node i in S do
        if (i.s = 'open')
            table_list.add(i.table_name + ' T' + i)
            for each condition c in i do
                condition_list.add('T' + i + '.' + c)
    for each edge j in S do
        if (j.e = 'present')
            if (j.right_node.s = 'open')
                join_list.add(
                    j.left_node + '.' +
                    j.left_attribute + ' = ' +
                    j.right_node + '.' +
                    j.right_attribute)
        else
            join_list.add(
                j.left_node + '.' +
                j.left_attribute + ' not in ' +
                translate_subgraph(subgraph(S, j.right_node), j.right_attribute))
    return 'select distinct' +
        ' T0.' + n0.primary_key +
        ' from ' + table_list +
        ' where ' + join_list +
        ' and ' + condition_list
```

Figura 3.8: Algoritmo `translate`.

las consultas SQL, hereda los problemas de eficiencia que presentan los sistemas basados en esta tecnología.

3.3.8. Cypher

Cypher es un lenguaje de consulta declarativo desarrollado específicamente para consultar la base de datos en grafo Neo4j que permite consultas expresivas y eficientes para obtener o actualizar la información almacenada. Consultas de patrones en grafos que habitualmente son complicadas en otros lenguajes resultan muy sencillas en Cypher [2], mostrando una alta capacidad expresiva. Combinando la sintaxis que proporciona para referenciar nodos, relaciones, y sus propiedades, podemos expresar patrones complejos de consulta [1]. Es posible acceder y modificar datos en Neo4j a través de consultas Cypher haciendo uso de diferentes canales, como son una API de Java y una API REST.

Cypher está diseñado para ser un lenguaje de consulta *humano*, cercano

```
translate_subgraph(S : Selection Graph, K : Key)

    table_list := ''
    condition_list := ''
    join_list := ''
    for each node i in S do
            table_list.add(i.table_name + 'T' + i)
            for each condition c in i do
                    condition_list.add( T' + i + '.' + c)
    for each edge j in S do
            join_list.add(
                    j.left_node + '.' +
                    j.left_attribute + '= ' +
                    j.right_node + '.' +
                    j.right_attribute)
    return 'select ' +
            ' T0.' + K +
            ' from ' + table_list +
            ' where ' + join_list +
            ' and ' + condition_list
```

Figura 3.9: Algoritmo `translate_subgraph`.

tanto para desarrolladores como para usuarios finales. Sus constructores usan palabras del inglés y la simbología utilizada en su sintaxis hace que las consultas se puedan interpretar muy fácilmente de manera visual. Además, comparte palabras clave, como `WHERE` y `ORDER BY`, con SQL, las expresiones para la detección de patrones se inspiran parcialmente en `SPARQL` y su semántica está inspirada en lenguajes como Haskell o Python.

La base de datos Neo4j sigue con mucha fidelidad el modelo de grafo (no hipergrafo) con propiedades, pero obliga que las aristas tengan un tipo asociado. En el contexto de los nodos, una etiqueta es una característica opcional que facilita su agrupación, y cada nodo puede tener un número indefinido de ellas. Además, Neo4j permite la creación de índices sobre propiedades de nodos que estén agrupados bajo la misma etiqueta con el fin de mejorar la eficiencia en los accesos. Por este motivo, es habitual crear índices para propiedades de nodos que habitualmente son utilizadas como restricciones de las consultas. Tras crear un índice, éste es manejado de manera automática e interna por Neo4j, actualizándolo cada vez que el grafo subyacente es modificado. Debido a que una base de datos como Neo4j está optimizada para acelerar las consultas cuyas restricciones están fundamentalmente basadas en relaciones, cuando las consultas imponen restricciones a través de propiedades en los nodos, el uso de índices puede mejorar considerablemente su rendimiento. Los índices imponen requerimientos adicionales de espacio y hacen más lentas las escrituras en la base de datos, por lo que no es recomendable crear demasiados. Decidir cuándo un índice es necesario no es una tarea trivial.

Por el hecho de ser un lenguaje declarativo, Cypher se centra en la claridad al expresar lo que se quiere obtener de un grafo, y no en cómo obtenerlo. Esta característica contrasta con Gremlin, que también puede trabajar sobre Neo4j pero tiene características imperativas y permite cierto control sobre las optimizaciones por parte del desarrollador de la consulta.

Al igual que hemos visto en los casos de SQL y SPARQL, en Cypher las consultas son construidas por concatenación de cláusulas, de forma que cada una proporciona resultados intermedios que son pasados a la cláusula siguiente. Las cláusulas más importantes en Cypher son:

- `MATCH`: Se utiliza para describir la estructura del patrón buscado, basado en relaciones.
- `WHERE`: Añade restricciones adicionales a los patrones.
- `RETURN`: Especifica qué datos deben ser devueltos.

Además de estas cláusulas, existen muchas otras en el lenguaje, como `COUNT` o `GROUP BY`. Cypher permite la asignación de variables identificadoras a cada uno de los elementos que se definen en un patrón (para tener la posibilidad de referenciarlos a lo largo de la consulta), restricciones sobre los elementos del patrón en la propia cláusula `MATCH`, así como una *representación gráfica* de las aristas en la propia consulta textual.

Un ejemplo de consulta en Cypher, que solicita los actores que participaron en la película Matrix, es:

```
MATCH (a:Actor)-[r:ACTED_IN]->(m:Movie{title:"The Matrix"}) RETURN a
```

En esta consulta se busca un patrón compuesto por dos nodos y una arista. Concretamente, un nodo asociado a la etiqueta `Actor` (identificado por `a`), un nodo asociado a la etiqueta `Movie` (identificado por `m`) y cuyo campo `title` tiene el valor `The Matrix`, y una arista (identificada por `r`) desde el nodo `a` hasta el nodo `m` de tipo `ACTED_IN`. Obsérvese que la restricción que exige la etiqueta asociada al segundo nodo es añadida en la propia cláusula `MATCH`, estas restricciones podrían haber sido incluidas en la cláusula `WHERE`, pero de esta forma la consulta es más expresiva.

La inclusión de Regular Pattern Matching en consultas Cypher es muy sencilla, tal y como muestra el siguiente ejemplo:

```
MATCH (a:Crew)-[:KNOWS|LOVES*..3]->(b:Crew) RETURN a,b
```

donde se muestra cómo Cypher permite especificar caminos de longitud va-

riable en los patrones, y añadir restricciones según los tipos que posean las aristas que lo forman. En este ejemplo se buscan los pares de elementos asociados a la etiqueta `Crew` que están conectados por un camino de longitud menor o igual que 3 y compuesto por aristas de tipo `KNOWS` o `LOVES`.

Además, para incrementar la modularidad y reducir la redundancia, Cypher permite asignar identificadores a los patrones. Por ejemplo:

```
p = (a:Crew)-[:KNOWS|LOVES*]->(b:Crew)
```

Posteriormente, podremos acceder a los detalles de un camino por medio de funciones auxiliares, por ejemplo: `nodes(p)` ($sop_V(p)$), `rels(p)` ($sop_E(p)$), o `length(p)` ($|p|$).

Pasamos ahora a mostrar una posible consulta Cypher que localiza todas las incidencias del patrón P_{ex} en un supuesto grafo almacenado en Neo4j (hemos convertido los valores numéricos de las etiquetas en P_{ex} a sus correspondientes cadenas de texto para respetar la sintaxis de Cypher):

```
MATCH (v3:ZERO)<-[e4:ONE]-(v2:ZERO)-[e2:ONE]->(v0:ZERO)-[e1:ZERO]->
   (v1:ONE)-[e3:ONE]->(v2)
WITH DISTINCT v0,v1,v2,v3,e1,e2,e3,e4
RETURN v0,v1,v2,v3
```

Nótese que se ha añadido la cláusula `WITH DISTINCT` para asegurar la inyectividad en la proyección (isomorfismo de subgrafos). En el caso de que al patrón buscado no se le exija la inyectividad con el subgrafo se podría prescindir de esta cláusula. Al igual que Gremlin (y a pesar de que no se han incluido en el resultado de la consulta), Cypher permite no sólo devolver los diferentes conjuntos de nodos que coinciden con el patrón, sino también las aristas (para ello sólo se deberían añadir los identificadores `e1,e2,e3` y `e4` a la cláusula `RETURN`). Esto es importante debido a que en muchas consultas de patrones en grafos no sólo se buscan los nodos que cumplen con el patrón sino también las aristas.

A diferencia con Gremlin, Cypher no es Turing completo, por lo que presenta algunas limitaciones. Por ejemplo, no es capaz de llevar a cabo algunos algoritmos de análisis de grafos, o no es capaz de expresar la forma en la que se quiere paralelizar una consulta. Para estos casos (igual que SQL) Cypher permite añadir funciones definidas por el usuario (extensiones del lenguaje) que hacen uso de la API de bajo nivel y alto rendimiento de Neo4j.

A pesar de las limitaciones, es un gran candidato para ser seleccionado como un estándar en consultas de bases de datos en grafo y, a pesar de que no es una solución ideada originalmente para convertirse en una propuesta de carácter

general, en los últimos tiempos se está convirtiendo en un estándar de facto como lenguaje de consulta a través del proyecto OpenCypher[14], entre otras cosas porque Cypher es un lenguaje relativamente sencillo pero muy potente.

Para consultas generales, Cypher es suficiente y es probablemente más rápido que Gremlin. La ventaja de Gremlin sobre Cypher es cuando se llega a un nivel alto de consultas de patrones. En Gremlin se puede definir mejor la forma en la que se debe recorrer el patrón (incluso se pueden definir algoritmos de recorrido propios), mientras que en Cypher esta tarea recae en el optimizador. Por ello, es habitual utilizar Cypher en situaciones habituales debido a su simplicidad, y utilizar Gremlin como un *fallback* cuando la expresividad de Cypher no es suficiente, o si se desea realizar una optimización manual de las consultas.

Las consultas sencillas en Cypher poseen un buen rendimiento, sin embargo no siempre es así cuando las consultas siguen patrones complejos. Esto ocurre porque, cuando hay condiciones múltiples, Cypher no permite que se le indique en qué orden aplicar dichas condiciones. Debe indicarse que a lo largo de las diferentes versiones que se han ido publicando de la base de datos Neo4j se han logrado grandes avances en cuanto a la eficiencia de las consultas Cypher, e incluso ya se pueden incluir consultas Cypher en una consulta Gremlin. Por ejemplo:

```
g.cypher("MATCH (a {name:'pedro'}) RETURN a").select("a").
    out("knows").name
```

Actualmente, debido a que Cypher todavía utiliza el mismo núcleo basado en JAVA que Blueprints (la misma API que utiliza Gremlin), no es más rápido que expresiones optimizadas escritas en el lenguaje imperativo subyacente. Sin embargo, según sus creadores, esto cambiará en el futuro debido al uso de estadísticas de consultas en la optimización, el indexado de registros, y la optimización transparente de datos basada en patrones usados, sin ni siquiera modificar la sintaxis actual de Cypher.

Con respecto a la potencia de Cypher en nuestro contexto, de manera natural este lenguaje permite consultas de patrones en grafos estructurales y semánticas, óptimas, exactas, y basadas en el isomorfismo de subgrafos. Permite un tipo de bounded simulation que, a pesar de que no permite definir el matching a través de relaciones en lugar de funciones, sí permite que las aristas en el patrón se proyecten sobre caminos del grafo. Además, se pueden imponer restricciones a esos caminos a través de expresiones que hacen uso del operador disyunción ($|$) y del cierre de Kleene ($*$). A través de funciones definidas por el usuario se podría extender Cypher para que permitiera el resto de tipo de consultas de

[14] `http://www.opencypher.org/`

patrones en grafos con propiedades presentadas en este capítulo.

Una limitación desde el punto de vista académico es que carece de un modelo formal asociado, y se ha construido con un carácter completamente aplicado, por lo que hay algunas formas de operar que no han sido validadas. A pesar de ello, por su excelente expresividad y aceptable rendimiento, y porque consume los datos de una base de datos en grafo muy extendida en su uso, con una versión de uso libre completa y con una gran comunidad de desarrolladores detrás, Cypher es el lenguaje de consulta de grafos que se ha elegido para implementar los diferentes modelos que se han desarrollado en esta tesis.

3.3.9. Otras Herramientas de Consulta

Acabaremos esta sección haciendo un breve repaso por algunas herramientas adicionales que nos han parecido interesantes y que pueden ayudar a entender la propuesta que daremos posteriormente.

En [18] proponen utilizar la Lógica Dinámica Proposicional (PDL) [81] como herramienta de consulta en bases de datos en grafo debido a que combina unas buenas propiedades de evaluación con una gran expresividad, proporcionando evaluaciones en tiempo polinomial, e incluso en tiempo lineal, para un gran número de patrones. Pero esta eficiencia tiene un coste, ya que solo permite trabajar con grafos dirigidos, con aristas etiquetadas, pero sin propiedades ni en nodos ni en aristas. Y lo que es más restrictivo, las expresiones en PDL deben ser acíclicas, es decir, no pueden expresar propiedades interesantes sobre ciclos en la estructura en grafo subyacente.

GraphQL[15] es un lenguaje de consulta declarativo desarrollado por la compañía Facebook para permitir el acceso a su información por parte de aplicaciones externas. Para ello, fue necesario desarrollar una API de consumo de datos lo suficientemente expresiva para describir todo el contenido almacenado en Facebook y lo suficientemente sencilla como para ser utilizada por la mayoría de desarrolladores. Las consultas en GraphQL tienen forma de árbol y poseen gran expresividad.

GraphFrames [62] es una librería de procesamiento de grafos para Apache Spark[16] que fue desarrollada por la UC de Berkeley en colaboración con el MIT. Está construida sobre el concepto de *DataFrame* de Spark, por lo que se beneficia de su escalabilidad y su alto rendimiento. Además, proporciona una API para poder manipular grafos haciendo uso de esta librería desde lenguajes como Python, JAVA o Scala. A partir de la versión 1.4, Spark incorpora GraphFrames

[15] `http://graphql.org/`

[16] `http://spark.apache.org/`

de forma nativa.

Graql[17] es un lenguaje de consulta declarativo orientado a grafos de conocimiento (un tipo de red semántica soportada por un modelo y un conjunto de reglas entre los datos) de tipo *Grakn* que utiliza métodos de razonamiento para obtener información tanto explícita (almacenada de manera directa en el grafo) como implícita (obtenida tras una serie de procedimientos automáticos a partir de los datos).

En [236] se presenta el lenguaje de consulta para grafos con propiedades PGQL, la intención es que el lenguaje posea las funcionalidades típicas de SQL extendiéndolo con características propias de las consultas en grafos: análisis de accesibilidad entre nodos, localización de caminos, y construcción de grafos. La sintaxis de PGQL es muy similar a la de SQL, y permite definir consultas de caminos con condiciones tanto en las etiquetas como en las propiedades (la sintaxis de consultas de caminos es muy similar a Cypher). A continuación se muestra un ejemplo de consulta PGQL que consulta el patrón P_{ex} en una base de datos (de nuevo, las etiquetas se presentan en formato de texto para no interferir con la sintaxis de PGQL):

```
SELECT v0.id, v1.id, v2.id, v3.id
FROM Graph
WHERE
(v0 WITH label = "ZERO") -[:ZERO]-> (v1),
(v1 WITH label = "ONE") -[:ONE]-> (v2),
(v2 WITH label = "ZERO") -[:ONE]-> (v3),
(v2) -[:ONE]-> (v0),
(v3 WITH label = "ZERO"),
v0.id != v1.id, v0.id != v2.id, v0.id != v3.id,
v1.id != v2.id, v1.id != v3.id, v2.id != v3.id
```

De nuevo, en la consulta presentada se exige que cada nodo del patrón se proyecte sobre un nodo diferente en el grafo (inyectividad). Si eliminásemos las dos últimas filas de condiciones permitiríamos que varios nodos del patrón se proyectasen sobre el mismo nodo en el grafo. Todas las consultas de patrones en grafos con propiedades que se pueden construir con PGQL pueden ser definidas a través de una consulta Cypher, haciendo que la potencia de ambos lenguajes sea equivalente. En cuanto a la expresividad, Cypher es más sencillo debido a que es capaz de ubicar el patrón y las restricciones en la misma cláusula. Las razones que han motivado el desarrollo de un lenguaje de consulta como PGQL es la baja eficiencia de determinadas consultas Cypher y el hecho de que éste

[17] `https://grakn.ai`

no posee un tipo de dato para modelar un objeto de tipo grafo.

En [132] se define un lenguaje de consulta y un motor para llevar a cabo Graph Pattern Matching que trabaja con un concepto de patrón similar al que proponemos en este trabajo: un *patrón* está compuesto por un grafo y un conjunto de predicados, que están definidos sobre los nodos en el patrón. Una coincidencia de un patrón en un grafo dado estará caracterizada por una proyección de los nodos del patrón a nodos del grafo (llamados los nodos *target* de la coincidencia), que verifica:

1. Cada nodo del patrón está asociado con un nodo del grafo del mismo tipo.
2. Cada arista del patrón está asociada con una arista del grafo del mismo tipo.
3. Todos los predicados se satisfacen.

Las condiciones 1 y 2 describen un homomorfismo de grafos tipados. En general este tipo de homomorfismo no requiere que la proyección sea inyectiva pero, si es necesario, es fácil añadir predicados para garantizar la inyectividad. Además, permiten el uso de un tipo de nodo/arista especial, que denotan con *ANY*, que permite que un nodo/arista en el patrón se asocie con un nodo/arista en el grafo de cualquier tipo.

3.4. Property Query Graph

A continuación presentamos los *Property Query Graphs* (PQG, para abreviar, a partir de ahora), nuestra propuesta para llevar a cabo consultas de patrones en grafos. Teniendo en cuenta las diversas clasificaciones apuntadas anteriormente, podemos decir que esta propuesta permite llevar a cabo consultas estructurales y semánticas, exactas, óptimas, y basadas en un tipo de Regular Pattern Matching que permite, además de proyectar aristas del patrón en caminos (no necesariamente aristas) que cumplan las restricciones impuestas, expresar restricciones más complejas sobre cada elemento del patrón y realizar consultas que posean ciclos.

Una de las características que buscamos en nuestra herramienta es que permita obtener (de alguna manera) patrones complementarios a un patrón dado. Esto significa que si una estructura no verifica un patrón debe verificar siempre uno de sus patrones complementarios. Como hemos visto en la sección anterior, muchas de las herramientas desarrolladas para llevar a cabo consultas de patrones en grafos exigen que se cumpla una proyección entre el patrón y la estructura

a evaluar. Dicha proyección impide evaluar la no existencia de elementos, algo que vamos a necesitar para generar estos patrones complementarios, por lo que nuestra propuesta no se basa en una proyección a la hora de verificar si una estructura cumple con un patrón determinado, sino que será construida en base a predicados lógicos, que facilitarán la generación de patrones complementarios.

Hemos de indicar que nuestro objetivo principal es el de proporcionar una formalización completa del modelo (frente a implementaciones incompletas desde el punto de vista formal, pero operativas), pero con el objetivo secundario de proporcionar una implementación que sea utilizable desde un punto de vista práctico (aunque más como una prueba de concepto que como una herramienta profesional en esta primera etapa).

En busca de nuestros objetivos, nos apoyaremos en el concepto de Grafo de Selección visto anteriormente, ampliándolo para añadirle Regular Pattern Matching y algunas características adicionales que nos permitirán obtener una mayor potencia expresiva en los patrones que se pueden construir.

Como principales características diferenciadoras respecto de los sistemas de consulta vistas en el apartado anterior, podemos indicar que:

- Los PQG pueden contener ciclos. Será un problema posterior considerar implementaciones de los PQG que manipulen los ciclos adecuadamente, considerar restricciones adicionales para asegurar ciertos niveles de eficiencia en su ejecución real, o preocuparse en la etapa de diseño de la consulta de crear un patrón que sea eficiente en la implementación disponible.

- Los PQG pueden evaluar subgrafos. Recordemos que en los Grafos de Selección clásicos sólo es posible evaluar un único nodo que representa a la tabla *target*. En el caso de los PQG, los *elementos fijos* (elementos que deben pertenecer al subgrafo bajo evaluación) serán representados a través de un predicado que obliga a que dichos elementos estén contenidos en el subgrafo a evaluar.

- Las aristas individuales del PQG pueden ser proyectadas sobre caminos en el grafo en el que se comprueba el patrón. Para ello se hará uso de predicados de forma similar a como se hace en Regular Pattern Matching.

- Los predicados asociados a nodos o aristas en el PQG pueden evaluar características estructurales y semánticas más allá de las propiedades almacenadas a través de la función μ (por ejemplo, a través de medidas como las definidas en el Capítulo 2).

Aunque ya hemos usado en la sección anterior la idea de que podemos disponer de un conjunto de predicados asociados a los elementos del patrón, vamos

a formalizar brevemente qué entendemos concretamente por un predicado definido sobre un grafo.

Tal y como muestra su definición, asociado a un grafo con propiedades tenemos una función μ que representa un conjunto de funciones (en particular, pueden ser predicados) asociados a nodos y aristas del grafo. Consideremos Θ, una colección de símbolos de función, predicados y constantes, que contiene todas las funciones de μ junto con constantes asociadas a cada elemento del grafo y, posiblemente, algunos símbolos adicionales, tanto de funciones como de predicados y constantes (por ejemplo, medidas definidas sobre los elementos del grafo). A partir de este conjunto de símbolos podemos definir un Lenguaje de Primer Orden con igualdad, L, haciendo uso de Θ como conjunto de símbolos no lógicos, sobre el que construimos, de la forma usual, el conjunto de términos del lenguaje y el conjunto de fórmulas, $FORM(L)$, que llamaremos *predicados*.

Aunque, en general, las fórmulas definibles en L se pueden aplicar a todos los objetos del universo, que en nuestro contexto estará compuesto por elementos de grafos (nodos, aristas, y estructuras formadas a partir de éstos), cuando queramos explicitar sobre qué tipos de objetos estamos trabajando en cada momento, podremos escribir $FORM_V(L)$ para indicar que son fórmulas aplicables sobre nodos, $FORM_E(L)$ para indicar que son fórmulas aplicables sobre aristas, $FORM_{\mathcal{P}}(L)$ para indicar que son fórmulas aplicables sobre caminos, etc.

En lo que sigue supondremos prefijado un Lenguaje sobre grafos, L, por lo que, con el objetivo de simplificar las expresiones que usemos, notaremos de forma general $FORM$ para denotar $FORM(L)$ cuando no haya posibilidad de confusión.

Además, y aprovechando la capacidad expresiva de los grafos generalizados, definimos las consultas sobre ellos haciendo uso de las mismas estructuras:

Definición 26. *Un* Property Query Graph (PQG) *sobre L es un grafo binario con propiedades sobre L, $Q = (V_Q, E_Q, \mu_Q)$, donde existen α y θ, propiedades destacadas en μ_Q, tales que:*

- $\alpha : V_Q \cup E_Q \to \{+, -\}$ *total.*
- $\theta : V_Q \cup E_Q \to FORM(L)$ *asocia un predicado binario, θ_x, a cada elemento x de $V_Q \cup E_Q$.*

Escribiremos $Q \in PQG(L)$ para denotar que Q es un Property Query Graph sobre L (si el lenguaje está prefijado y no hay posibilidad de confusión, escribiremos simplemente $Q \in PQG$).

El sentido de usar predicados binarios es que en la semántica asociada a un PQG usaremos la segunda entrada de estos predicados para poder hablar de

condiciones de pertenencia sobre subgrafos de G (el grafo general sobre el que estamos evaluando las consultas), mientras que la primera esperará recibir como entrada elementos adecuados al tipo de elemento al que está asociado. Así, si S es un subgrafo y $a \in V_Q$ entonces $\theta_a(., S) \in FORM_V$, y si $e \in E_Q$ entonces $\theta_e(., S) \in FORM_{\mathcal{P}}$. Por ejemplo:

$$\theta_a(v, S) = \exists z \in S \ (z \rightsquigarrow v)$$
$$\theta_e(\rho, S) = \exists y, z(y \overset{\rho}{\rightsquigarrow} z \wedge y \notin S \wedge z \in S)$$

El primer predicado tendrá sentido para nodos, y se verificará cuando exista un camino en G que conecta un nodo de S (el subgrafo que estamos evaluando) con v, el nodo de entrada sobre el que se evalúa. El segundo predicado tendrá sentido para caminos, y se verificará cuando el camino evaluado, ρ, conecta S con su complementario (en G).

Dado un PQG en las condiciones anteriores, notaremos x^+, respectivamente x^-, para indicar que $\alpha(x) = +$, respectivamente $\alpha(x) = -$, y V_Q^+/V_Q^- (respectivamente, E_Q^+/E_Q^-) el conjunto de nodos (respectivamente, aristas) positivos/negativos. Si para un elemento x, θ_x no está explícitamente definida, supondremos que θ_x es una tautología, que podemos denotar en general por T.

Tal y como veremos a continuación, intuitivamente, los elementos positivos del patrón representan elementos que deben estar presentes en el grafo sobre el que se realiza la consulta y que verifican los predicados asociados, mientras que los elementos negativos en el patrón representan elementos que no deben estar presentes en el grafo.

Para poder expresar con más facilidad las condiciones necesarias que definen la aplicación de un PQG sobre un grafo, así como los resultados que veremos en lo que queda de capítulo, introducimos a continuación una serie de notaciones que generan predicados aplicables sobre elementos del grafo:

Definición 27. *Dado $Q = (V_Q, E_Q, \mu_Q)$ un PQG, el conjunto de Q-predicados asociados a Q es:*

1. *Para cada arista, $e \in E_Q$, definimos los Q-predicados asociados como:*

$$Q_{e^o}(v, S) = \exists \rho \in \mathcal{P}_v^o(G) \ \left(\theta_e(\rho, S) \wedge \theta_{e^o}(\rho^o, S) \wedge \theta_{e^i}(\rho^i, S)\right)$$
$$Q_{e^i}(v, S) = \exists \rho \in \mathcal{P}_v^i(G) \ \left(\theta_e(\rho, S) \wedge \theta_{e^o}(\rho^o, S) \wedge \theta_{e^i}(\rho^i, S)\right)$$

En general, escribiremos $Q_{e^*}(v, S)$, donde $* \in \{o, i\}$, y notaremos:

$$Q_{e^*}^+ = Q_{e^*}, \qquad Q_{e^*}^- = \neg Q_{e^*}$$

2. Para cada nodo, $n \in V_Q$, definimos el Q-predicado asociado como:

$$Q_n(S) = \exists v \in V \left(\bigwedge_{e \in \gamma^o(n)} Q_{e^o}^{\alpha(e)}(v, S) \ \wedge \bigwedge_{e \in \gamma^i(n)} Q_{e^i}^{\alpha(e)}(v, S) \right)$$
$$= \exists v \in V \left(\bigwedge_{e \in \gamma^*(n)} Q_{e^*}^{\alpha(e)}(v, S) \right)$$

Y que podemos escribir en general como:

$$Q_n(S) = \exists v \in V \left(\bigwedge_{e \in \gamma(n)} Q_e^{\alpha(e)}(v, S) \right)$$

ya que para cada nodo no hay posibilidad de confusión. Además, notaremos:

$$Q_n^+ = Q_n, \qquad Q_n^- = \neg Q_n$$

A partir de estas notaciones, podemos definir formalmente cuándo un subgrafo verifica un PQG determinado:

Definición 28. *Dado un subgrafo S de un grafo con propiedades, $G = (V, E, \mu)$, y un Property Query Graph, $Q = (V_Q, E_Q, \mu_Q)$, ambos sobre el lenguaje L, diremos que S* verifica *Q, y lo denotaremos $S \vDash Q$, si se verifica la fórmula:*

$$Q(S) = \bigwedge_{n \in V_Q} Q_n^{\alpha(n)}(S)$$

En caso contrario, escribiremos: $S \nvDash Q$.

En la Figura 3.10 se muestra un PQG genérico a modo de ejemplo.

Uno de los objetivos que persiguen los PQG es proporcionar la capacidad expresiva suficiente para expresar condiciones que hacen uso de elementos que están fuera del subgrafo que se está evaluando, algo que se ha demostrado necesario para disponer de un lenguaje de consultas potente y que no está presente en las soluciones vistas anteriormente en este capítulo.

Obsérvese que, en particular, usando $S = G$ podemos definir cuándo un grafo verifica un PQG.

Aunque la definición de PQG que hemos presentado hace uso de grafos binarios (no hipergrafos), ya que proyecta aristas sobre caminos que conectan pares

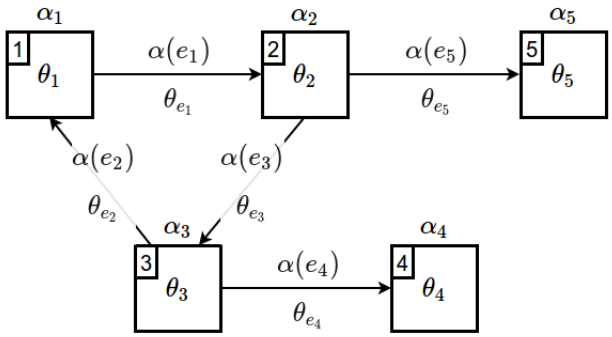


Figura 3.10: Ejemplo de Property Query Graph.

de nodos, el concepto de grafo generalizado es suficientemente flexible como para permitir otras interpretaciones en las que se pueden considerar PQGs que hagan uso de estructuras más generales. Además, y es importante resaltar este hecho, aunque un PQG sea binario, puede aplicarse sobre grafos con propiedades que no lo sean (es decir, G podría ser un hipergrafo generalizado), ya que el concepto de camino que conecta pares de nodos se define independientemente de la aridad de las aristas que intervienen. En estos casos, se debería usar una notación algo más compleja para poder definir los Q-predicados, pero es completamente factible. Por motivos de simplicidad, y por la falta de bases de datos de hipergrafos, hemos restringido las definiciones presentadas a estos casos particulares, pero quedan abiertas para ser extendidas a los casos más generales en el momento en el que el uso de hipergrafos se generalice como medio de modelado y almacenamiento, ya que en las pocas ocasiones en que se han precisado siempre se ha resuelto el problema por medio de la creación de nuevos tipos de nodos y aristas binarias que simulan la presencia de hiperaristas.

Antes de pasar a analizar algunas propiedades interesantes sobre los PQG y la forma de construirlos, veamos algunos ejemplos que permitan entender cómo se interpretan y qué capacidad expresiva permiten.

3.4.1. Ejemplos de Property Query Graphs

A lo largo de este parágrafo, y a modo de ejemplo, presentaremos una colección de pequeños PQG sobre un grafo con propiedades concreto con el objeto de mostrar la forma en que funcionan y su capacidad expresiva.

En la Figura 3.11 se presenta un grafo con propiedades que se corresponde con una sección de una base de datos basada en grafos que contiene información acerca de los personajes principales de la serie Starwars y que es utilizada frecuentemente como ejemplo sencillo para hacer demostraciones relacionadas con

las capacidades de las bases de datos en grafo [18]. En lo que sigue haremos uso de este grafo para presentar algunos patrones que hagan uso del lenguaje sobre el que está definido y para comprobar su verificación sobre algunos subgrafos concretos del mismo.

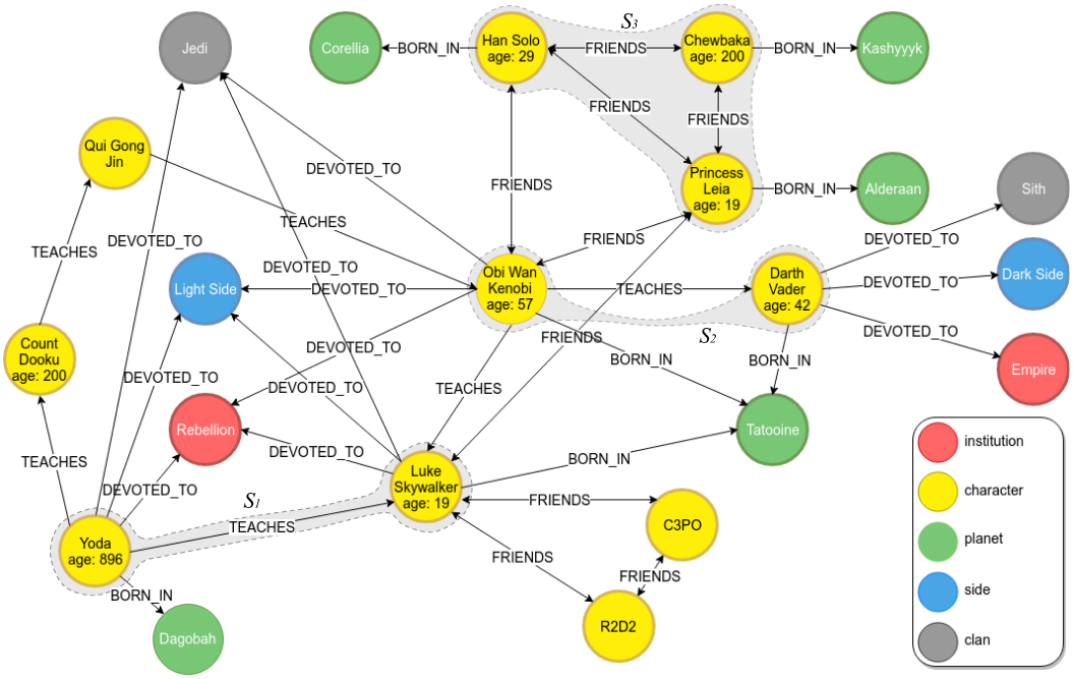


Figura 3.11: Grafo Starwars para ilustrar ejemplos de Property Query Graphs.

Con el fin de simplificar la representación de consultas y subgrafos, una de las propiedades en μ, a la que denominaremos τ, será expresada directamente sobre la aristas. En el caso de los nodos, la propiedad τ será representada a través de colores. Estas dos propiedades representan una clasificación de tipo sobre nodos y aristas. Además, la propiedad *name* de los nodos será representada directamente sobre los mismos, y las aristas no dirigidas, que conectan dos nodos pero no imponen un orden concreto en la relación, serán representadas como aristas bidireccionales.

La representación gráfica de los PQG de ejemplo se muestran en las figuras 3.12 a 3.17. Cuando analicemos la interpretación de estas consultas, también indicaremos algunos subgrafos de G que los verifican. Cada elemento en estos PQG tiene asociada la representación de su propiedad α directamente por medio de un símbolo $+/-$, y de su propiedad θ directamente en el elemento (si el predicado asociado a un elemento en un PQG es una tautología, dicho predicado no será representado). En expresiones del tipo $\tau(\rho) = X$ en el predicado de una

[18] http://console.neo4j.org/?id=StarWars

arista, X será interpretada como una expresión regular que debe ser cumplida por la secuencia de propiedades τ de las aristas en el camino ρ.

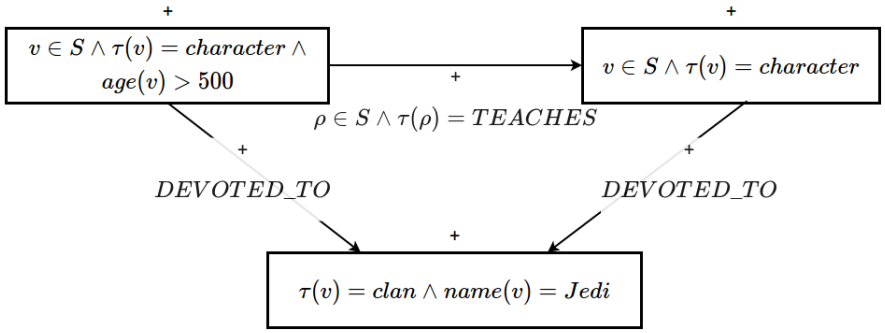


Figura 3.12: Ejemplo 1 PQG.

El PQG P_1 (Figura 3.12) se puede interpretar en lenguaje natural a través de la siguiente sentencia: *Personajes y relación alumno-maestro en la que ambos son devotos de los Jedi y el maestro tenga más de 500 años.* En este caso se imponen restricciones estructurales a través de la presencia de aristas y a través de predicados que hacen uso de atributos (τ, *name*, y *age*). Este PQG se verificará en subgrafos en los que puedan ser proyectados dos nodos y una arista que los une (los tres elementos marcados como elementos positivos en el subgrafo) que cumplan con las restricciones impuestas. En el caso de que existiera un personaje que se haya enseñado a sí mismo (que posea un lazo de tipo `TEACHES`) que tenga más de 500 años y sea devoto de los Jedi, un subgrafo que contenga este nodo también verificaría este patrón. El subgrafo marcado como S_1 en la Figura 3.11 verifica P_1.

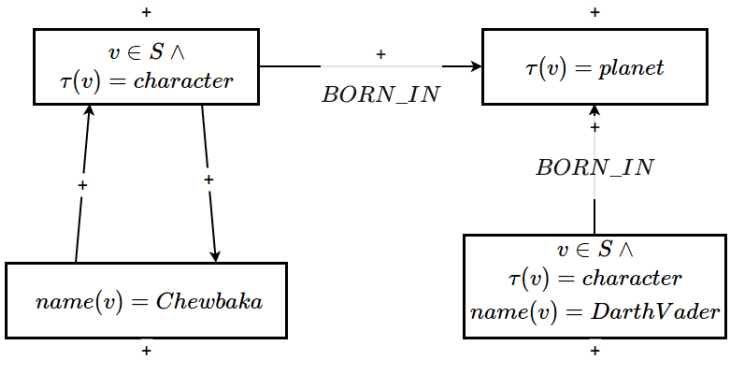


Figura 3.13: Ejemplo 2 PQG.

El PQG P_2 (Figura 3.13) se puede interpretar en lenguaje natural a través de la siguiente sentencia: *Darth Vader y personaje que provenga del mismo planeta que él y que posea un camino que lo conecte con Chewbaka.* Este PQG

presenta un nodo positivo que representa al personaje *Chewbaka* (por medio de la propiedad *name*) e impone una restricción que exige que uno de los nodos en el subgrafo evaluado esté conectado con él. Un subgrafo que verifica P_2 es el subgrafo destacado como S_2 en la Figura 3.11.

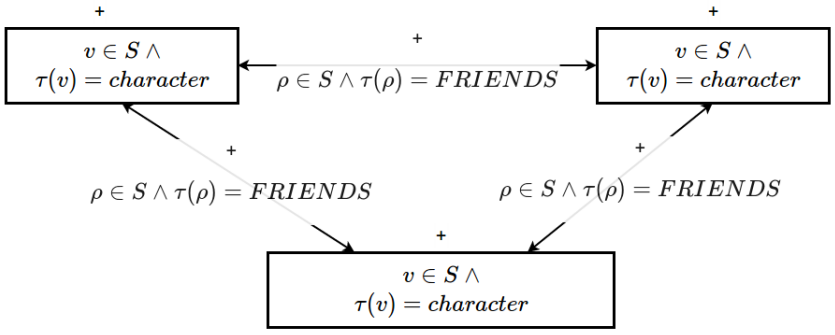


Figura 3.14: Ejemplo 3 PQG.

El PQG P_3 (Figura 3.14) presenta un ciclo a través de relaciones de amistad, y S_3 (subgrafo resaltado en la Figura 3.11) es un subgrafo que lo verifica. Cualquier subgrafo que contenga tres personajes que son amigos entre sí (existen relaciones bidirecciones de tipo `FRIENDS` entre ellos) verificará P_3. Por ejemplo, subgrafos que contengan el ciclo formado por `Luke Skywalker`, `R2D2` y `C3PO`, o el ciclo formado por `Han Solo`, `Princess Leia` y `Chewbaka`.

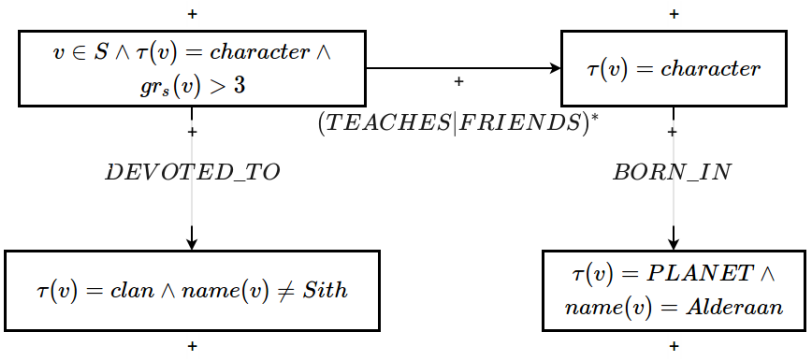


Figura 3.15: Ejemplo 4 PQG.

El PQG P_4 (Figura 3.15) puede ser interpretado en lenguaje natural a través de la siguiente sentencia: *Personaje que esté conectado a través de relaciones de tipo FRIENDS o TEACHES con alguien que provenga de Alderaan, que tenga grado de salida superior a tres, y que sea devoto de un clan que no sean los Sith.* En este caso, se ha utilizado una expresión regular para expresar un camino compuesto de relaciones de tipo `FRIEND` o `TEACHES`. Además, se ha

usado una función auxiliar, $gr_s(v)$, para referirse al grado de salida del nodo v. Cualquier subgrafo conteniendo el nodo `Luke Skywalker` o el nodo `Obi Wan Kenobi` verificará P_4.

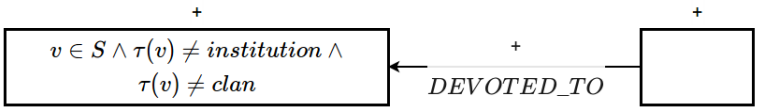


Figura 3.16: Ejemplo 5 PQG.

El PQG P_5 (Figura 3.16) representa la consulta: *Nodos que no sean instituciones ni clanes, pero tengan devotos*, y sólo es verificado por los nodos de tipo *side*. En este caso, se ha utilizado un nodo cuya propiedad θ es una tautología (que, como indicamos, ha sido representado a través de un nodo vacío).

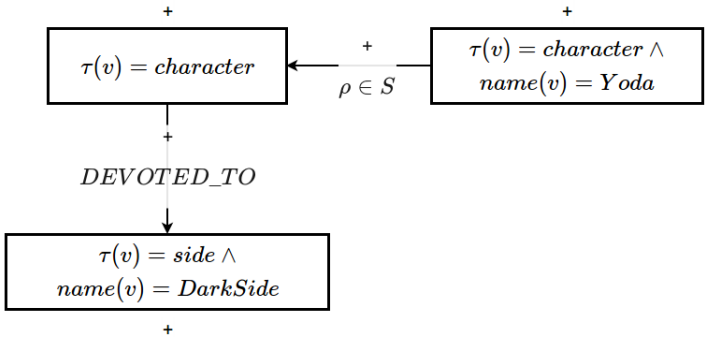


Figura 3.17: Ejemplo 6 PQG.

El PQG P_6 (Figura 3.17) podría ser interpretado a través de la siguiente sentencia: *Caminos que relacionen a Yoda con personajes del Lado Oscuro*. Cualquier subgrafo que contenga el camino (`Yoda`) $\rightarrow$ (`Count Dooku`) $\rightarrow$ (`Qui Gong Jin`) $\rightarrow$ (`Obi Wan Kenobi`) $\rightarrow$ (`Darth Vader`) verificará P_6.

3.4.2. Conjuntos de refinamiento

En la sección anterior hemos visto que los PQG se pueden interpretar como predicados sobre la familia de subgrafos de un grafo prefijado G. Sería interesante obtener formas computacionalmente efectivas de construir PQG a partir de operaciones básicas para obtener familias de predicados que nos permitan analizar la estructura de los subgrafos de G de una forma automática.

A continuación vamos a dar una primera aproximación a un método constructivo que sea de utilidad para realizar este tipo de tareas sobre un grafo.

Aunque no se proporcione un aparato completo desde el punto de vista lógico, veremos en el siguiente capítulo que es lo suficientemente potente como para permitirnos resolver algunos interesantes problemas de clasificación automática en grafos.

Comenzamos dando una definición natural cuando se trabaja con consultas sobre estructuras, y que nos permite ver cuándo un PQG tiene más capacidad que otro para discriminar entre subgrafos.

Definición 29. *Dados* $Q_1,\ Q_2 \in PQG$*, diremos que* Q_1 refina Q_2 *en* G*, y lo notaremos como* $Q_1 \preceq_G Q_2$ *(escribiremos, simplemente,* $\preceq$ *cuando trabajemos sobre un grafo* G *prefijado) si:*

$$\forall S \subseteq G\ (S \vDash Q_1 \Rightarrow S \vDash Q_2)$$

Teniendo en cuenta que los PQG pueden ser tratados como predicados, también lo podremos escribir como:

$$\forall S \subseteq G\ (S \vDash Q_1 \rightarrow Q_2)$$

o también:

$$\vDash_G Q_1 \rightarrow Q_2$$

En la Figura 3.18 se muestra un ejemplo de un PQG que refina a otro. En este caso, el PQG de la izquierda refina al PQG situado a la derecha ya que, además de exigir que uno de los nodos en el subgrafo a evaluar esté conectado con un nodo que no pertenece a dicho subgrafo a través de una relación de tipo `publish`, también se exige que el nodo destino de dicha relación posea una arista entrante que parta de un nodo que no pertenece al subgrafo bajo evaluación.

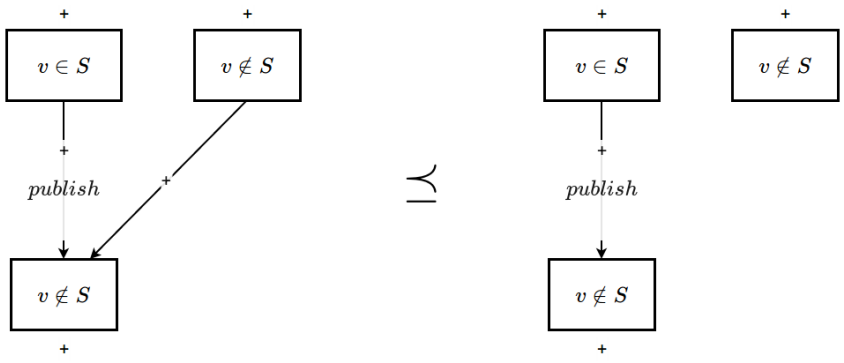


Figura 3.18: Refinado entre PQG.

De forma natural, podemos definir cuándo dos PQG son equivalentes como consultas, que se tendrá cuando los dos verifiquen exactamente los mismos subgrafos.

Definición 30. *Dados* $Q_1,\ Q_2 \in PQG$*, diremos que son* equivalentes *en* G*, y lo notaremos como* $Q_1 \equiv_G Q_2$ *(usaremos, simplemente,* $\equiv$ *cuando trabajemos sobre un grafo* G *prefijado) si:*

$$Q_1 \preceq_G Q_2 \ \wedge \ Q_2 \preceq_G Q_1$$

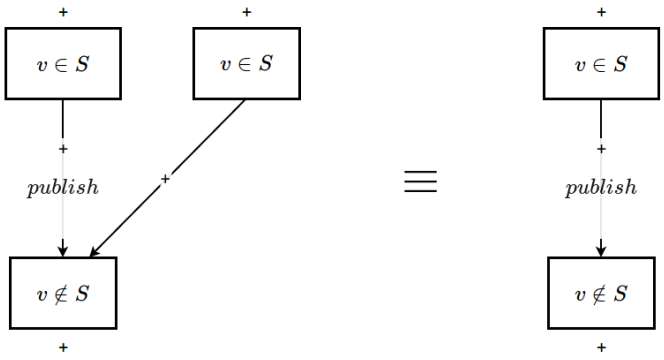


Figura 3.19: Equivalencia de PQG.

En la Figura 3.19 se muestra un ejemplo de dos PQG que son equivalentes. Nótese que la única diferencia estructural del PQG de la izquierda con respecto al de la derecha es la existencia de un nodo perteneciente al subgrafo a evaluar que posea una arista de salida cuyo destino sea un nodo que no esté incluido en el subgrafo a evaluar. Dicha restricción está incluida en el PQG de la derecha, ya que de existir un nodo en el subgrafo bajo evaluación con una arista de tipo `publish` conectada con un nodo que no pertenezca a dicho subgrafo, también existe una arista con las mismas restricciones excepto el tipo impuesto a la arista.

Es fácil probar el siguiente resultado, que nos dice que $\preceq_G$ genera una relación de orden sobre los PQG considerando la equivalencia como igualdad (o lo que es lo mismo, trabajando en el espacio cociente que determina la equivalencia).

Teorema 6. *Para todo Grafo con Propiedades,* G*, se tiene que* $(PQG, \preceq_G)$ *es un conjunto ordenado. Es decir:*

1. $\forall Q \in PQG\ (Q \preceq_G Q)$.
2. $Q_1 \preceq_G Q_2 \ \wedge \ Q_2 \preceq_G Q_1 \Rightarrow Q1 \equiv_G Q2$.
3. $Q_1 \preceq_G Q_2 \ \wedge \ Q_2 \preceq_G Q_3 \Rightarrow Q_1 \preceq_G Q_3$.

Es fácil comprobar que, en general, $\preceq_G$ no genera una relación de orden total. Para ello, basta encontrar dos PQG, Q_1 y Q_2, para los cuales no se cumpla $(Q_1 \to Q_2) \vee (Q_2 \to Q_1)$. Por ejemplo, si Q_1 es un PQG compuesto por un único nodo positivo con la restricción $v \in S$, y Q_2 es un PQG compuesto también por un único nodo positivo con la restricción $v \notin S$, entonces Q_1 exigirá que el subgrafo bajo evaluación no esté vacío, y Q_2 exigirá que exista algún nodo en el grafo que no pertenezca al subgrafo bajo evaluación. Ambas restricciones son independientes por lo que no existe implicación entre los predicados que los PQG representan.

Vamos a analizar la relación existente entre la estructura topológica de un PQG y su funcionalidad como predicado sobre subgrafos. En general, es un problema complicado intentar extraer propiedades lógicas del predicado a partir de las propiedades estructurales del grafo que lo representa, pero podemos obtener algunas condiciones útiles que nos permitirán manipular constructivamente las estructuras para modificar la interpretación de los PQG de forma controlada.

Definición 31. *Dados Q_1, $Q_2 \in PQG$, diremos que Q_1 es una* extensión Q^--conservativa *de Q_2, y lo notaremos como $Q_2 \subseteq^- Q_1$, si:*

1. *$Q_2 \subseteq Q_1$ (como grafos con propiedades, por lo que en los elementos de Q_2 coinciden los valores de α y θ para ambos PQG).*

2. *Para cada nodo negativo de Q_2, $n \in V^-_{Q_2}$, y cada arista incidente en él en Q_1, $e \in \gamma_{Q_1}(n)$, existe una arista incidente en él en Q_2, $e' \in \gamma_{Q_2}(n)$, que impone la misma restricción, es decir: $Q_e \equiv Q_{e'}$.*

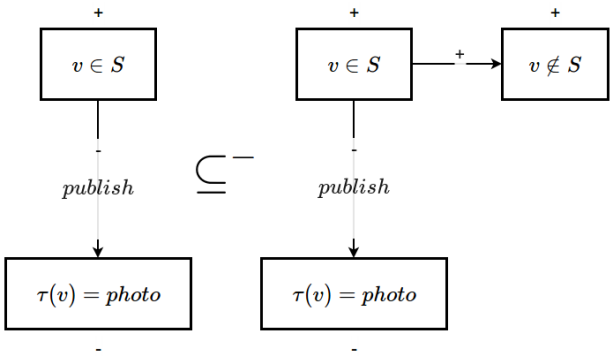


Figura 3.20: Extensión Q^--conservativa.

En la Figura 3.20 se muestra un ejemplo de una extensión Q^--conservativa. La extensión realizada en el PQG de la izquierda para obtener el PQG de la derecha, ha impuesto nuevas restricciones sobre el nodo positivo, pero no ha añadido nuevas restricciones al nodo negativo.

Como los nodos negativos añaden restricciones a la verificación de subgrafos, las extensiones Q^--conservativas aseguran que no estamos añadiendo restricciones adicionales a éstos(añadiendo más información a las aristas incidentes en los nodos negativos), por lo que podemos dar el siguiente resultado:

Teorema 7. *Dados $Q_1, Q_2 \in PQG$, si $Q_2 \subseteq^- Q_1$ entonces $Q_1 \preceq Q_2$.*

Demostración. Debido a que los Q-predicados para aristas dependen exclusivamente de la información en la propia arista (que considera el valor de θ en sus extremos, sin importar el valor de α en los mismos), podemos afirmar que:

$$\forall e \in E_{Q_2}\ ({Q_1}_e^{\alpha(e)} = {Q_2}_e^{\alpha(e)})$$

Teniendo en cuenta este hecho, vamos a analizar cómo se comportan los Q-predicados asociados a los nodos para ambos PQG:

- Si $n \in V_{Q_2}^-$, debido a que $Q_2 \subseteq^- Q_1$, es inmediato que ${Q_1}_n^- = {Q_2}_n^-$ (es la misma fórmula, no solo son equivalentes).
- Si $n \in V_{Q_2}^+$, entonces ${Q_1}_n^+ \rightarrow {Q_2}_n^+$, ya que (notaremos por γ_1, γ_2 las funciones de incidencia de Q_1 y Q_2, respectivamente):

$$\begin{aligned}
{Q_1}_n^+ &= \exists v \in V\ \left(\bigwedge_{e \in \gamma_1(n)} {Q_1}_e^{\alpha(e)} \right) \\
&= \exists v \in V\ \left(\bigwedge_{e \in \gamma_1(n) \cap E_{Q_2}} {Q_1}_e^{\alpha(e)} \ \wedge \bigwedge_{e \in \gamma_1(n) \setminus E_{Q_2}} {Q_1}_e^{\alpha(e)} \right) \\
&= \exists v \in V\ \left(\bigwedge_{e \in \gamma_2(n) \cap E_{Q_2}} {Q_2}_e^{\alpha(e)} \ \wedge \bigwedge_{e \in \gamma_1(n) \setminus E_{Q_2}} {Q_1}_e^{\alpha(e)} \right) \\
&\rightarrow {Q_2}_n^+
\end{aligned}$$

En consecuencia:

$$\begin{aligned}
Q_1 = \bigwedge_{n \in V_{Q_1}} {Q_1}_n^{\alpha(n)} &= \bigwedge_{n \in V_{Q_2}} {Q_1}_n^{\alpha(n)} \ \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} {Q_1}_n^{\alpha(n)} \\
&= \bigwedge_{n \in V_{Q_2}^+} {Q_1}_n^{\alpha(n)} \ \wedge \bigwedge_{n \in V_{Q_2}^-} {Q_1}_n^{\alpha(n)} \ \wedge \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} {Q_1}_n^{\alpha(n)}
\end{aligned}$$

$$\begin{aligned}
&\rightarrow \bigwedge_{n \in V_{Q_2}^+} {Q_2}_n^{\alpha(n)} \ \wedge \ \bigwedge_{n \in V_{Q_2}^-} {Q_2}_n^{\alpha(n)} \ \wedge \ \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} {Q_1}_n^{\alpha(n)} \\
&= \bigwedge_{n \in V_{Q_2}} {Q_2}_n^{\alpha(n)} \ \wedge \ \bigwedge_{n \in V_{Q_1} \setminus V_{Q_2}} {Q_1}_n^{\alpha(n)} \\
&\rightarrow Q_2
\end{aligned}$$

□

El resultado anterior sugiere que se puede refinar un PQG añadiendo nodos (de cualquier signo) y aristas a los nodos positivos ya existentes, pero debido a la interpretación (negada) de los Q-predicados asociados a nodos negativos, hay que tener la precaución de mantener el entorno de los mismos para estar seguros de que añadir más aristas a ellos no debilita las condiciones impuestas a los subgrafos evaluados (y, por tanto, no conseguiríamos predicados que refinan).

Con el fin de dar métodos controlados de generación de consultas, en lo que sigue daremos un método constructivo para ir refinando un PQG por pasos unitarios. Para ello, comenzaremos viendo cómo se comportan los PQG cuando se clonan nodos.

Un clon consiste en hacer copias de nodos existentes, clonando todas las aristas incidentes en ellos (y entre ellos, en caso de que clonemos varios nodos que están conectados en el PQG original). Por supuesto, la operación de clonación se puede hacer sobre grafos con propiedades cualesquiera, y así la presentamos.

Definición 32. *Dado $G = (V, E, \mu)$ un grafo con propiedades, y $W \subseteq V$, definimos el* clon de G por duplicación de W, *y lo notaremos por Cl_G^W, como el grafo con propiedades siguiente:*

$$Cl_G^W = (V \cup W', \ E \cup E', \mu \cup \{(n', \mu(n))\}_{n \in W} \cup \{(e', \mu(e))\}_{e' \in E'})$$

donde:

- *para cada $n \in W$, n' es un nodo nuevo, $W' = \{n' \ : \ n \in W\}$, y*
- *E' es un conjunto de aristas nuevas que se consiguen a partir de las aristas incidentes en nodos de W donde se sustituyen de todas las formas posibles los nodos de W por copias de W' (de forma que aparecen aristas clonadas que conectan nodos originales con nodos copia, y también aristas clonadas que conectan nodos copia).*

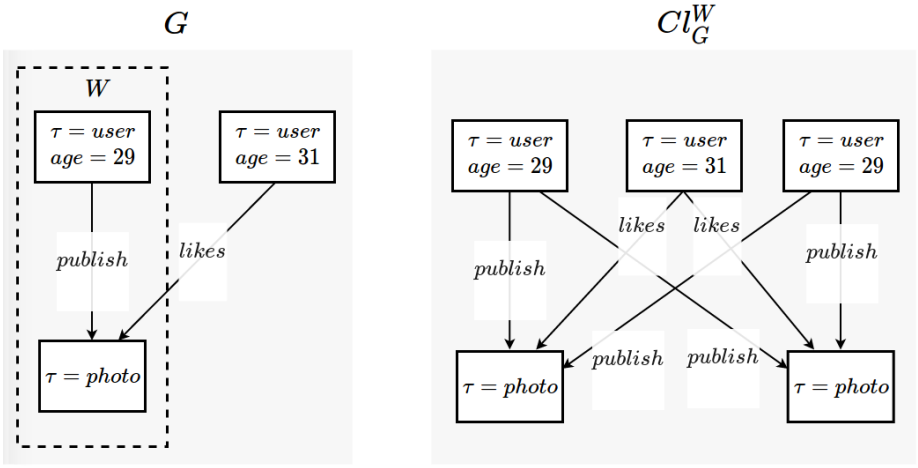


Figura 3.21: Clon de un grafo.

En la Figura 3.21 se muestra un ejemplo de un grafo clonado por dupliación de dos de sus nodos. En el grafo original a la izquierda se han marcado los dos nodos a ser clonados. El resultado de la clonación se observa en el grafo a la derecha.

El siguiente resultado nos indica que la clonación de nodos positivos no altera la interpretación de las consultas.

Teorema 8. *Si $Q \in PQG$ y $W \subseteq V_Q^+$, entonces $Cl_Q^W \equiv Q$.*

Demostración. Para facilitar la notación, sea $Q_1 = Cl_Q^W$. Entonces, siguiendo un razonamiento similar al de la demostración anterior:

$$\begin{aligned}
Q_1 &= \bigwedge_{n \in V_{Q_1}} {Q_1}_n^{\alpha(n)} \\
&= \bigwedge_{n \in V_Q} {Q_1}_n^{\alpha(n)} \wedge \bigwedge_{n \in W} {Q_1}_{n'}^{\alpha(n')} \\
&= \bigwedge_{n \in V_Q \setminus \gamma_Q(W)} {Q_1}_n^{\alpha(n)} \wedge \bigwedge_{n \in \gamma_Q(W)} {Q_1}_n^{\alpha(n)} \wedge \bigwedge_{n \in W} {Q_1}_{n'}^{\alpha(n')} \\
&= \bigwedge_{n \in V_Q \setminus \gamma_Q(W)} Q_n^{\alpha(n)} \wedge \bigwedge_{n \in \gamma_Q(W)} Q_n^{\alpha(n)} \wedge \bigwedge_{n \in W} Q_n^{\alpha(n)} \\
&= Q
\end{aligned}$$

□

Siguiendo con la idea de obtener herramientas que nos permitan construir PQG de manera automática, el concepto de *refinamiento* que introducimos

a continuación completa las operaciones que podemos hacer para refinar un PQG. En cierta forma, un conjunto de refinamiento forma una partición por refinamientos de un PQG dado.

Definición 33. *Dado $Q \in PQG$. Diremos que $R \subseteq PQG$ es un* conjunto de refinamiento *de Q en G si verifica:*

1. $\forall\ Q' \in R\ (Q' \preceq_G Q)$
2. $\forall\ S \subseteq G\ (S \vDash Q \Rightarrow \exists!\ Q' \in R\ (S \vDash Q'))$

Teniendo en cuenta que los PQG se pueden interpretar como funciones que toman subgrafos y devuelven 0, cuando $S \nvDash Q$, o 1, cuando $S \vDash Q$, un conjunto de refinamiento para Q es una conjunto de PQG que verifica:

$$Q = \sum_{Q' \in R} Q'$$

Estamos ya en condiciones de dar algunos conjuntos de refinamiento que nos permitirán automatizar los procesos de creación y modificación de Property Query Graphs. Comenzaremos por la operación más sencilla, que consiste en ver de qué formas se pueden añadir nuevos nodos a un PQG existente:

Teorema 9 (Añadir nodo nuevo a Q)**.** *Dado $Q \in PQG$ y $m \notin V_Q$, entonces el conjunto que notaremos como $Q + \{m\}$, formado por:*

$$Q_1 = (V_Q \cup \{m\},\ E_Q,\ \alpha_Q \cup (m, +),\ \theta_Q \cup (m, T))$$
$$Q_2 = (V_Q \cup \{m\},\ E_Q,\ \alpha_Q \cup (m, -),\ \theta_Q \cup (m, T))$$

es un conjunto de refinamiento de Q en G.

Demostración. Hemos de comprobar que se verifican las dos condiciones necesarias para que sea un conjunto de refinamiento:

1. Es evidente que $Q \subseteq^- Q_1$ y $Q \subseteq^- Q_2$, por lo que $Q_1 \preceq Q$ y $Q_2 \preceq Q$.
2. Sea $S \subseteq G$ tal que $S \vDash Q$. Tenemos que:

   $$Q_1 = Q\ \wedge\ Q_m$$
   $$Q_2 = Q\ \wedge\ \neg Q_m$$

   donde $Q_m = \exists v \in V\ (T)$.

   Si $G \neq \emptyset$, entonces $S \vDash Q_1$ y $S \nvDash Q_2$.

   Si $G = \emptyset$, entonces $S \nvDash Q_1$ y $S \vDash Q_2$.

□

Como norma general, $G \neq \emptyset$, por lo que esta operación realmente no refina, en el sentido de que $Q1 \equiv Q$ y $Q2 \equiv \neg T$. Sin embargo, a pesar de que obtenemos un PQG equivalente, esta operación es muy útil para añadir nuevos nodos a un PQG a los que posteriormente se le podrán ir añadiendo nuevas restricciones. En la Figura 3.22 se muestra un diagrama explicativo de este primer conjunto de refinamiento.

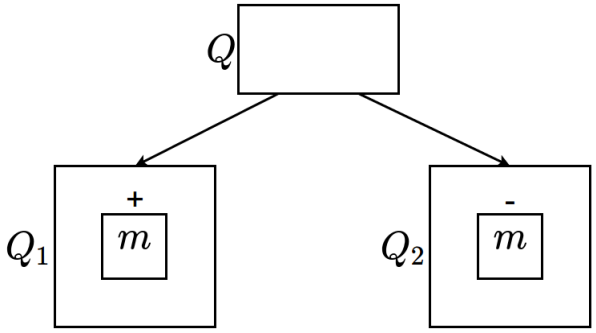


Figura 3.22: Refinamiento añadir nodo.

Teniendo en cuenta los resultados anteriores que daban relaciones entre las propiedades estructurales del PQG y su interpretación semántica como consulta, pasamos a dar un segundo conjunto de refinamiento que nos indica cómo interviene la creación de aristas entre nodos existentes. Para mantener que todos refinen al PQG original, hemos de restringir la adición de aristas a los nodos positivos.

Teorema 10 (Añadir arista nueva entre nodos positivos de Q)**.** *Dado $Q \in PQG$ y $n, m \in V_Q^+$, entonces el conjunto que denotaremos como $Q + \{n^+ \xrightarrow{e^*} m^+\}$ ($* \in \{+, -\}$), formado por (donde $Q' = Cl_Q^{\{n,m\}}$):*

$$Q_1 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e^*} m^+\},\ \theta_{Q'} \cup (e, T))$$
$$Q_2 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e^*} m^-\},\ \theta_{Q'} \cup (e, T))$$
$$Q_3 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e^*} m^+\},\ \theta_{Q'} \cup (e, T))$$
$$Q_4 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e^*} m^-\},\ \theta_{Q'} \cup (e, T))$$

es un conjunto de refinamiento de Q en G.

Demostración.

1. Como Q' es un clon de Q, y $\{n, m\} \subseteq V_Q^+$, tenemos que $Q \equiv Q'$. Además, por construcción, $Q' \subseteq^- Q_1, Q_2, Q_3, Q_4$, por lo que $Q_1, Q_2, Q_3, Q_4 \preceq Q' \equiv Q$.

2. Consideremos los predicados:

$$P_n = \exists v \in V \left(\bigwedge_{a \in \gamma(n)} Q_a^{\alpha(a)} \ \wedge Q_{e^o}^{\alpha(e)} \right)$$

$$P_m = \exists v \in V \left(\bigwedge_{a \in \gamma(m)} Q_a^{\alpha(a)} \ \wedge Q_{e^i}^{\alpha(e)} \right)$$

   Si $S \vDash Q_n$ y $S \vDash Q_m$, entonces tenemos 4 opciones mutuamente excluyentes, según se verifique $S \vDash P_n$ y/o $S \vDash P_m$, que son:

   - $S \vDash P_n \ \wedge \ S \vDash P_m \Rightarrow S \vDash Q_1$
   - $S \vDash P_n \ \wedge \ S \nvDash P_m \Rightarrow S \vDash Q_2$
   - $S \nvDash P_n \ \wedge \ S \vDash P_m \Rightarrow S \vDash Q_3$
   - $S \nvDash P_n \ \wedge \ S \nvDash P_m \Rightarrow S \vDash Q_4$

□

En la Figura 3.23 se muestra un diagrama explicativo de este conjunto de refinamiento. Si $n = m$ (la arista añadida es un lazo), entonces el conjunto de refinamiento anterior queda reducido a dos PQG, los equivalentes a Q_1 y Q_4.

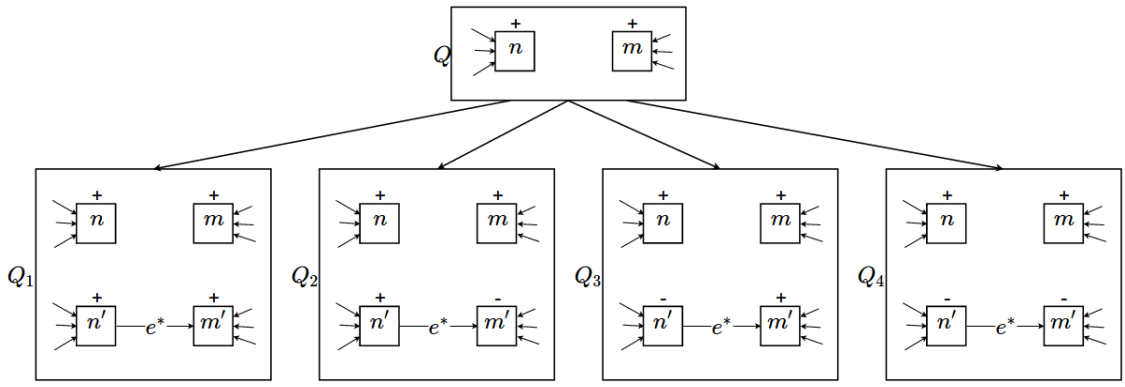


Figura 3.23: Refinamiento añadir arista.

La siguiente modificación necesaria es la de añadir un predicado adicional a una arista existente. Para mantener las condiciones estructurales necesarias, restringimos esta operación a las aristas positivas que conectan nodos positivos.

Teorema 11 (Añadir predicado a arista positiva entre nodos positivos de Q)**.** *Dado $Q \in PQG$ $n, m \in V_Q^+$, con $n^+ \xrightarrow{e^+} m^+$, y $\varphi \in FORM(L)$, el conjunto que notaremos como $Q + \{n^+ \xrightarrow{e \wedge \varphi} m^+\}$, formado por (donde $Q' = Cl_Q^{\{n,m\}}$):*

$$Q1 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e'} m^+\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q2 = (V_{Q'},\ E_{Q'} \cup \{n^+ \xrightarrow{e'} m^-\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q3 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e'} m^+\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$
$$Q4 = (V_{Q'},\ E_{Q'} \cup \{n^- \xrightarrow{e'} m^-\},\ \theta_{Q'} \cup (e', \theta_e \wedge \varphi))$$

es un conjunto de refinamiento de Q en G.

Demostración. La demostración es similar a la realizada en los casos anteriores. □

En la Figura 3.24 se muestra un diagrama explicativo de este conjunto de refinamiento.

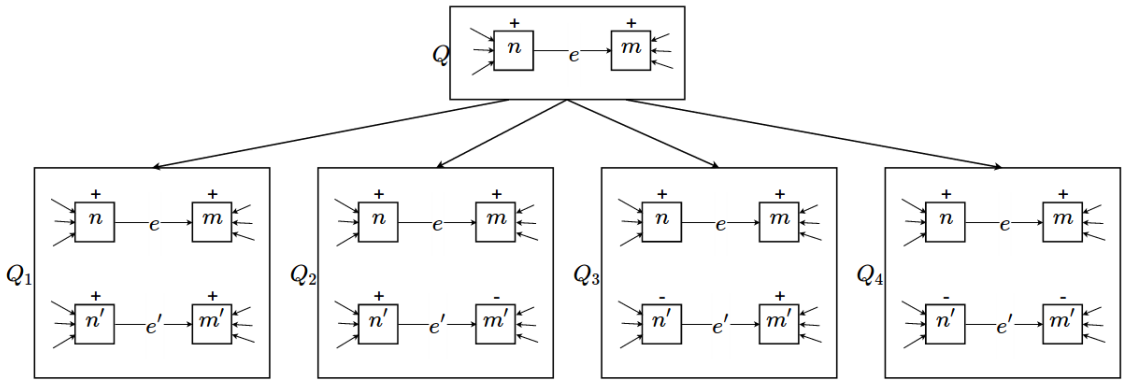


Figura 3.24: Refinamiento añadir predicado a arista.

Por último, la modificación que nos queda es la de añadir predicados a nodos existentes. De nuevo, hemos de restringir esta operación a los casos que no plantean problemas, cuando los nodos afectados son positivos (el nodo al que se añade el predicado, y los conectados a él).

Teorema 12 (Añadir predicado a nodo positivo con entorno positivo en Q)**.** *Dado $Q \in PQG$, $n \in V_Q^+$, con $\mathcal{N}_Q(n) \subseteq V_Q^+$, y $\varphi \in FORM(L)$. Definimos el conjunto que denotaremos como $Q + \{n \wedge \varphi\}$ formado por:*

$$\{Q_\sigma = (V_{Q'}, E_{Q'}, \alpha_{Q'} \cup \sigma, \theta_{Q'} \cup (n', \theta_n \wedge \varphi)) \ :\ \sigma \in \{+,-\}^{\mathcal{N}_Q(n)}\}$$

donde $Q' = Cl_Q^{\mathcal{N}_Q(n)}$, *y* $\{+,-\}^{\mathcal{N}_Q(n)}$ *es el conjunto todas las posibles asignaciones de signo a los elementos de* $\mathcal{N}_Q(n)$ *(el entorno, en* Q*, del nodo* n*).*

Entonces $Q + \{n \wedge \varphi\}$ *es un conjunto de refinamiento de* Q *en* G*.*

Demostración. La demostración es similar a la realizada en los casos anteriores. Solo hay que tener en cuenta que cuando se modifica el nodo n no solo queda modificado el Q-predicado asociado a él sino también el de todos sus nodos adyacentes.

Por ello, el procedimiento que se ha seguido para cubrir todas las posibles opciones de asignación de signos para los nodos involucrados es por medio del conjunto de funciones $\{+,-\}^{\mathcal{N}_Q(n)}$ (recordemos que en $\mathcal{N}_Q(n)$ también se tiene en cuenta el centro, n). □

En la Figura 3.25 se muestra un diagrama explicativo de este conjunto de refinamiento.

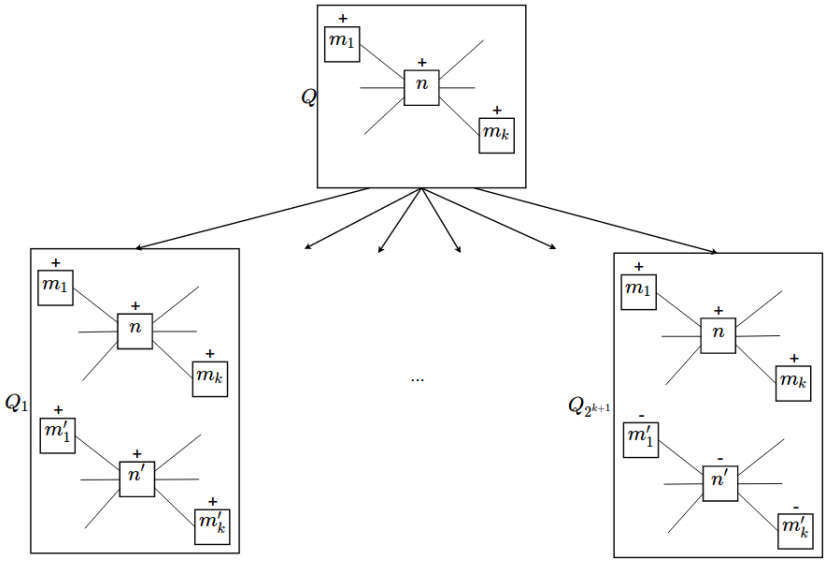


Figura 3.25: Refinamiento añadir predicado a nodo.

Se debe tener en cuenta que los refinamientos anteriores generan estructuras que pueden ser simplificadas. A continuación vamos a definir la operación principal que, cuando se puede aplicar, permite simplificar un PQG determinado obteniendo otro equivalente con menor número de elementos.

Definición 34. *Dado* $Q \in PQG$*, diremos que* $Q' \subseteq Q$ *es* redundante *en* Q *si* $Q \equiv Q - Q'$*. Donde* $Q - Q'$ *es el subgrafo de* Q *dado por:*

$$(V_Q \setminus V_{Q'}, E_Q \setminus (E_{Q'} \cup \{\gamma(n) : \ n \in V_{Q'}\}), \mu_Q)$$

Veamos un primer resultado que, analizando nodos, nos permite obtener versiones simplificadas de un PQG por medio de la eliminación de nodos redundantes positivos:

Teorema 13. *Sea $Q \in PQG$, y $n \in V_Q^+$ tal que existe $m \in V_Q$ verificando:*

- $\alpha(n) = \alpha(m)$, $\theta_n \equiv \theta_m$.
- *Para cada $e \in \gamma(n)$, existe $e' \in \gamma(m)$, verificando $\alpha(e) = \alpha(e')$, $\theta_e = \theta_{e'}$ y $\gamma(e) \setminus \{n\} = \gamma(e') \setminus \{m\}$.*

Entonces, n es redundante en Q.

Esencialmente, la condición que impone el resultado anterior es que m sea un clon de n pero, posiblemente, con más aristas conectadas. Teniendo en mente esta idea intuitiva, la prueba es directa a partir de las condiciones impuestas.

Podemos obtener un resultado similar para aristas por medio del siguiente resultado:

Teorema 14. *Sea $Q \in PQG$, y dos aristas, $e, e' \in E_Q$, tales que $n^+ \xrightarrow{e} m^+$ y $n^+ \xrightarrow{e'} m^+$. Si $\theta_e \to \theta_{e'}$ entonces e' es redundante en Q.*

A partir de los resultados anteriores podemos dar versiones simplificadas de los conjuntos de refinamiento vistos, agrupando nodos positivos y aristas positivas en aquellos casos en los que, tras la clonación inicial, el signo del elemento duplicado se ha mantenido con el original, así como en los casos en los que el signo se ha mantenido y se ha añadido un predicado adicional. En las Figuras 3.26 a 3.28 se muestran diagramas de los conjuntos de refinamiento $Q + \{n \wedge \varphi\}$, $Q + \{n^+ \xrightarrow{e \wedge \varphi} m^+\}$ y $Q + \{n \wedge \varphi\}$ aplicando las simplificaciones presentadas.

Por ejemplo, para construir el patrón P_5 una posibilidad sería seguir la siguiente secuencia de refinamientos:

$$\begin{aligned}
Q_1 &= Q_\emptyset + \{n_1\} \\
Q_2 &= Q_1 + \{n_1 \wedge (v \in S \wedge \tau(v) \neq \texttt{institution} \wedge \tau(v) \neq \texttt{clan}\} \\
Q_3 &= Q_2 + \{n_2\} \\
Q_4 &= Q_3 + \{n_2 \xrightarrow{e_1} n_1\} \\
P_5 &= Q_4 + \{n_2 \xrightarrow{e_1 \wedge (\tau(\rho) = \texttt{DEVOTED_TO})} n_1\}
\end{aligned}$$

El proceso de construcción seguido por esta secuencia de refinamientos es presentado en la Figura 3.29.

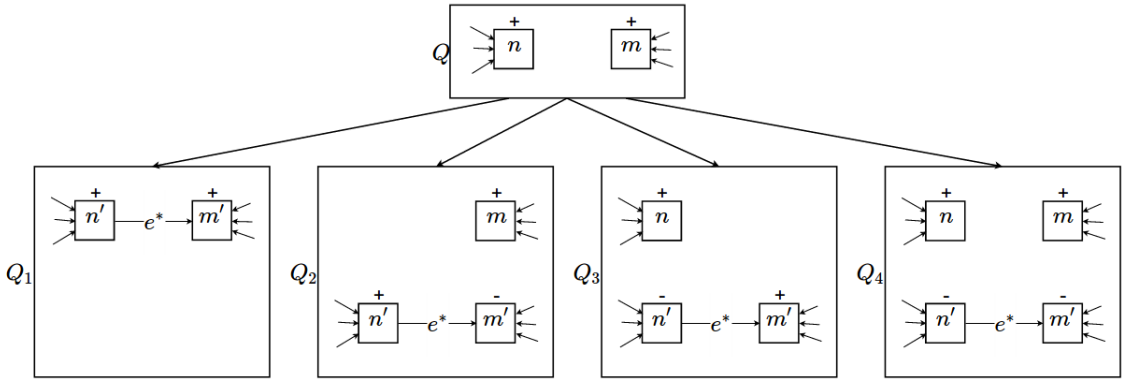


Figura 3.26: Refinamiento añadir arista (simplificado).

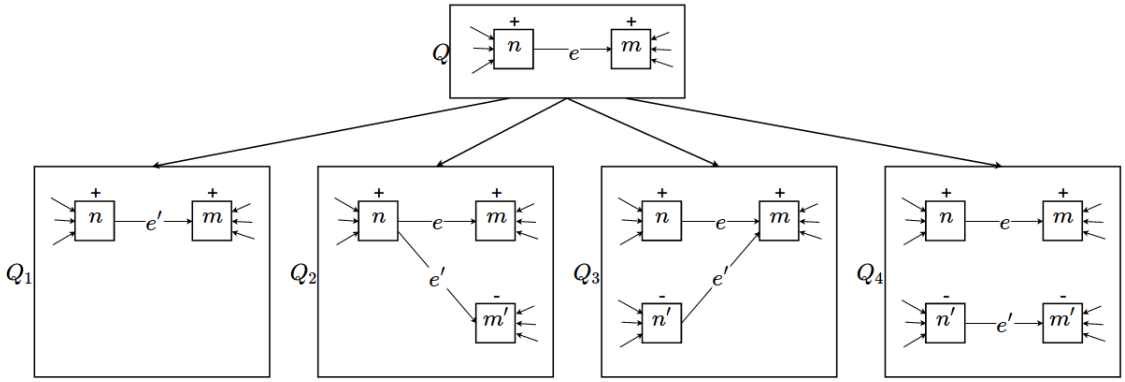


Figura 3.27: Refinamiento añadir predicado a arista (simplificado).

A partir de la estructura como grafo de un PQG no es fácil obtener un PQG complementario con él. Sin embargo, como veremos en el capítulo siguiente, hay muchos procesos de análisis sobre grafos con propiedades en los que necesitamos trabajar con sucesiones de consultas que verifiquen algunas propiedades de contención y complementariedad como predicados. Los refinamientos vistos en esta sección vienen a cubrir esta carencia. De esta forma, por medio de refinamientos, podemos construir un árbol de particiones encajadas que tiene los nodos etiquetados de la siguiente forma (Figura 3.30):

- El nodo raíz está etiquetado con Q_0 (un PQG inicial cualquiera).
- Si un nodo del árbol está etiquetado con Q, y $R = (Q_1, \ldots, Q_n)$ es un conjunto de refinamiento de Q, entonces sus nodos hijos estarán etiquetados por los elementos de R.

Obsérvese que la construcción del árbol anterior depende por completo de la elección de conjuntos de refinamiento que se elija en cada ramificación.

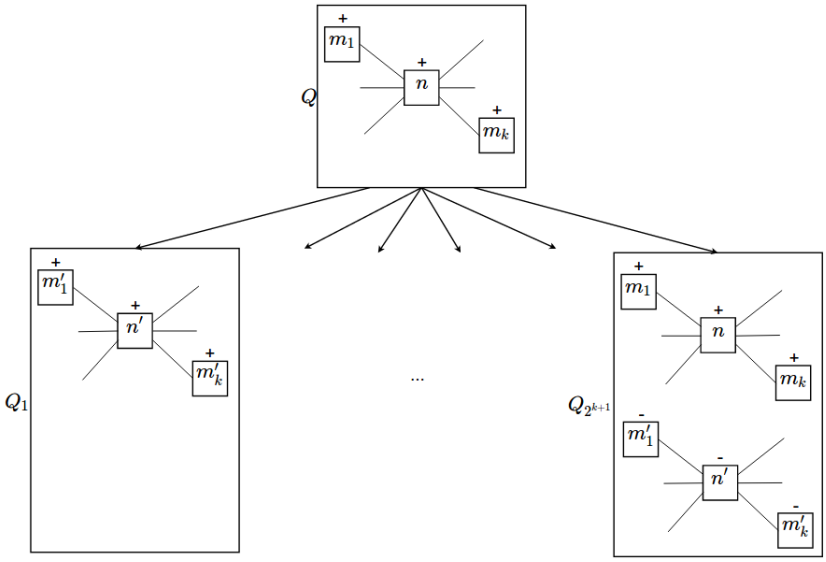


Figura 3.28: Refinamiento añadir predicado a nodo (simplificado).

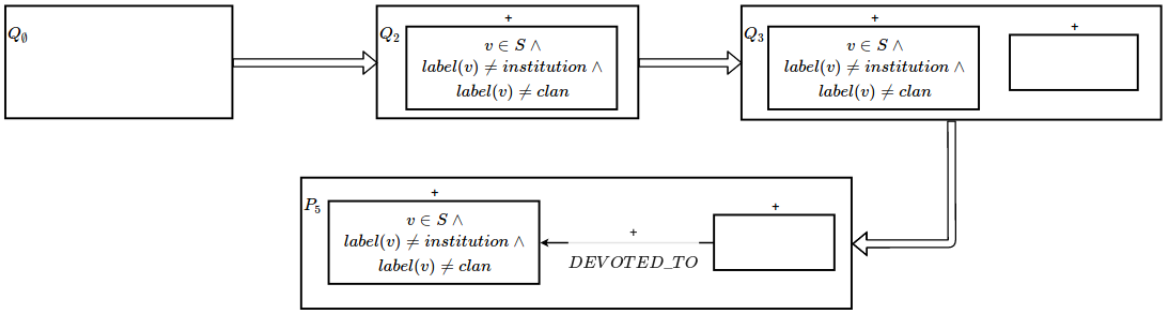


Figura 3.29: Sucesión de refinamientos.

Los refinamientos que hemos presentado en los resultados anteriores son una opción, pero no es la única posible. Por ejemplo, se pueden considerar refinamientos que, en vez de añadir restricciones a elementos positivos, aligere las impuestas por los elementos negativos, consiguiendo nuevos PQG que refinan al anterior, y usando la adición de predicados por medio de la disyunción en vez de la conjunción.

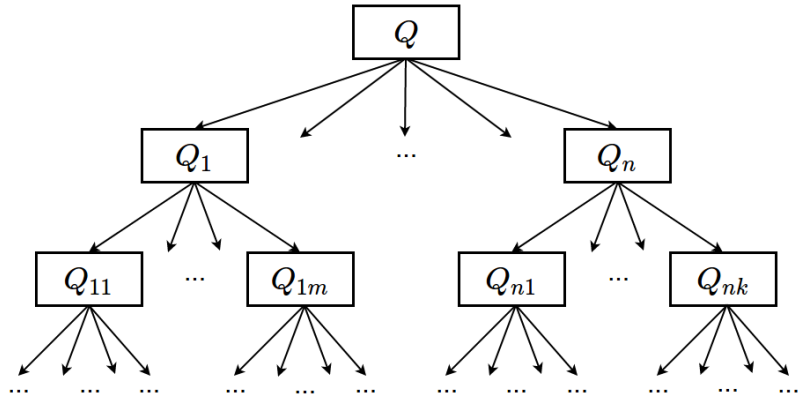


Figura 3.30: Árbol de refinamientos.

Capítulo 4

Árboles de decisión para grafos con propiedades

Como ya comentamos en el capítulo de Fundamentos, un árbol de decisión es un modelo de clasificación (y regresión) que, a partir de las características de un objeto dado, y aplicando una serie de reglas, es capaz de asociar una clase a dicho objeto (o un valor continuo en el caso de regresión). La inducción de árboles de decisión a partir de un conjunto de objetos previamente clasificados es uno de los modelos de aprendizaje automático más populares debido, entre otras cosas, a la baja demanda computacional en su entrenamiento y a la facilidad de interpretación de sus resultados por parte de los humanos.

La generación automática de árboles de decisión a partir de conjuntos de datos es una tarea muy útil en muchas áreas del conocimiento ya que nos permite predecir características de nuevos objetos de manera automática a partir de un conjunto de ejemplos dado [203, 210]. En este capítulo analizaremos diferentes técnicas que permiten la construcción automática de árboles de decisión a partir de datos estructurados en forma de grafos tal y como hemos visto en capítulos anteriores. Algunas de ellas provienen de metodologías que están orientadas a la detección de patrones frecuentes en grandes conjuntos de datos (y que habitualmente se enmarcan dentro de la Minería de Datos [138]), otros, por el contrario, provienen de metodologías orientadas a la generalización de comportamientos (y que, en consecuencia, se enmarcan más naturalmente dentro del Aprendizaje Automático). Veremos que algunas de las metodologías que analizaremos inducen árboles de decisión que permiten clasificar automáticamente elementos de un grafo (nodos, aristas, o subgrafos) según sus características locales, mientras que otros inducen árboles de decisión que permiten clasificar automáticamente grafos completos según las características generales del mismo.

La mayoría de los modelos de aprendizaje automático poseen la limitación de que están preparados para trabajar con datos no relacionales, es decir, objetos con una serie de atributos propios, sin tener en cuenta los atributos de otros objetos con los que están relacionados y que también pueden aportar información sobre los mismos. Los algoritmos de aprendizaje automático que tienen en cuenta las relaciones existentes entre las diversas muestras del conjunto de entrenamiento (y los atributos de otros objetos con los que el objeto a predecir está relacionado) caen dentro de las denominadas *Tecnologías de Aprendizaje Automático Relacionales* (Multi-Relacionales en caso de que se tenga en cuenta más de un tipo de relación en la estructura de los datos [140]), un área de investigación emergente en los últimos años [206].

El sistema FOIL [187] (First-Order Inductive Learner), desarrollado por Quinlan et al. en la década de los 90, permite generar automáticamente reglas de clasificación en lógica de primer orden para datos relacionales. Permitiendo a FOIL utilizar predicados binarios o de aridad superior que representan relaciones entre conceptos, este puede ser considerado el primer sistema de aprendizaje en datos relacionales para tareas de clasificación. En 1998, Blockeel y De Readt presentaron el algoritmo TILDE [33], que construye árboles de decisión con lógica de primer orden utilizando el algoritmo C4.5 [188] (una extensión del famoso ID3 presentado también por Quinlan para la creación automática de árboles de decisión) y que permite el aprendizaje relacional a través de una orientación similar a la seguida por FOIL. En 1999, Knobbe et al. propusieron el algoritmo MRDTL [129] (Multi-Relational Decision Tree Learning) como una mejora del algoritmo TILDE [33] orientado a la clasificación de registros en una base de datos relacional. Los algoritmos para construcción de árboles de decisión clásicos [185], árboles de decisión proposicionales [187], árboles de decisión con lógica de primer orden [33], y árboles de decisión multi-relacionales [129] son idénticos en su estructura: los nodos internos contienen tests, y los nodos hoja contienen los valores a predecir.

Aunque el algoritmo MRDTL ya fue nombrado en el Capítulo 3 en el contexto de los sistemas de consulta en grafos, aquí lo presentaremos completo. A pesar de que MRDTL y TILDE se basan en la misma idea, el algoritmo TILDE utiliza predicados de primer orden para representar los nodos en el árbol de decisión, mientras que MRDTL utiliza el concepto de grafo de selección que ya hemos analizado [70].

Existen otras técnicas que persiguen objetivos similares. Por ejemplo, Graph-Based Induction (GBI) es una técnica de Minería de Datos relacional que extrae de manera eficiente patrones frecuentes de datos estructurados en forma de grafo a través de la unión de pares de nodos conectados, y Decisión Tree Graph-Based Induction (DT-GBI) es un algoritmo de construcción de árboles de decisión para datos estructurados en forma de grafo utilizando los principios de GBI [87].

La diferencia principal entre los algoritmos anteriores es que aquellos que construyen árboles de decisión a partir de datos estructurados en forma de grafo pertenecientes a la familia de MRDTL están orientados a construir árboles clasificadores de nodos, mientras que los métodos basados en GBI están orientados a construir árboles clasificadores de grafos completos.

La propuesta de inducción automática de árboles clasificadores a partir de grafos con propiedades que presentaremos aquí tiene como objetivo proporcionar un marco que permita clasificar cualquier tipo de estructura en un grafo, desde simples nodos (como en MRDTL) hasta aristas, caminos o cualquier otra estructura multi-relacional, haciendo uso del concepto de Property Query Graph presentado en el capítulo anterior.

En todos los algoritmos de aprendizaje automático relacionales el principal cuello de botella para mejorar su eficiencia es el espacio de hipótesis. Puede haber una gran cantidad de atributos relacionados con la estructura a clasificar que no proporcionan información relevante a la clasificación buscada. Al tener en cuenta las relaciones, el espacio de búsqueda se hace más grande, lo que conlleva una pérdida de eficiencia de los algoritmos habituales. Como veremos más adelante, el algoritmo MRDTL-2 [140] lucha contra este problema por medio de la reducción del espacio de búsqueda limitando los posibles refinamientos que se pueden realizar sobre un grafo de selección. Siguiendo una estrategia similar, nuestra propuesta permitirá acotar los posibles refinamientos que se permiten aplicar a un Property Query Graph para la construcción de los diversos nodos interiores del árbol de decisión.

En este capítulo comenzaremos profundizando en el concepto de árbol de decisión, que se presentó superficialmente en el capítulo 2, y analizaremos los algoritmos más representativos que hacen uso de los mismos y que han servido como base para muchos desarrollos posteriores. A continuación realizaremos un repaso por las diferentes técnicas existentes de inducción de árboles de decisión relacionales, y presentaremos nuestra propuesta basada en el uso de Property Query Graphs como herramienta de evaluación en los nodos del árbol. Una vez planteada nuestra propuesta, presentaremos algunos ejemplos de aplicación de la misma.

4.1. Árboles de Decisión

Los árboles de decisión constituyen, quizás, uno de los modelos de clasificación más conocidos. Originalmente se diseñó en el marco de la Teoría de la Decisión y la Estadística, pero más adelante se posicionó como un buen modelo dentro del aprendizaje automático [203]. Aunque al principio se diseñaron

como modelos de clasificación, también existen árboles de regresión (es el caso del algoritmo M5 [84] que hace uso de modelos de regresión lineal en los nodos del árbol construido), así como algoritmos para construirlos automáticamente a partir de datos, donde el objetivo es construir el árbol que mejor aproxime la salida a predecir.

Dada la larga historia y el gran interés en los árboles de decisión, existen numerosas y buenas referencias que los estudian en profundidad [210, 161, 131]. Remitimos al lector al capítulo 2, donde se presentan brevemente los fundamentos de los árboles de decisión.

Algunas de las características que hacen de los árboles de decisión un buen método de aprendizaje son:

- *Sencillo de entender e interpretar:* Los humanos son capaces de entender los árboles de decisión directamente al contemplar su estructura, si no contiene una cantidad excesiva de nodos y niveles. Además, es sencillo convertir un árbol de decisión en un conjunto de reglas implementable en cualquier lenguaje de programación.

- *Requiere poca preparación de datos:* Otras técnicas a menudo requieren de procesos de normalización de datos y/o la construcción de nuevas variables a partir de las originales, así como otras tareas de preparación de datos previa a la construcción del modelo.

- *Capaz de manejar datos numéricos y categóricos:* Otras técnicas suelen estar especializadas en el análisis de conjuntos de muestras que presentan un único tipo de dato. (Por ejemplo, determinados sistemas de reglas pueden usarse con datos lógicos, mientras que las redes neuronales sólo pueden usarse con datos numéricos). Los árboles de decisión se adaptan con facilidad a cualquier tipo de dato en los atributos de los objetos.

- *Utiliza un modelo de caja blanca:* Si una situación dada es observable en una muestra del modelo, el camino seguido por la muestra en su evaluación por el árbol proporciona una explicación acerca de su clasificación. Por el contrario, en un modelo de caja negra (por ejemplo, una red neuronal artificial) la relación entre el comportamiento interno del modelo y el resultado ofrecido es mucho más difusa.

- *Posibilidad de ser validado mediante pruebas estadísticas:* Permite estudiar la fiabilidad del modelo.

- *Robusto:* Puede funcionar bien aunque haya sido construido en presencia de ruido.

- *Funciona bien con conjuntos de datos grandes:* Pueden ser construidos de manera automática a partir de grandes cantidades de datos utilizando una cantidad de recursos razonable.

Durante los últimos 30 años se han desarrollado numerosos algoritmos que construyen árboles de decisión de manera automática a partir de un conjunto de datos de ejemplo [160, 121, **?**]. Aunque no es el único posible, de entre todos ellos probablemente ID3 sea el más representativo debido a que constituye la base de la mayoría de algoritmos de este tipo desarrollados posteriormente.

Por ello, y debido a que el método que usaremos para nuestra propuesta se basa completamente en este famoso algoritmo, pasamos a presentar sus fundamentos, así como diferentes mejoras que han sido desarrolladas posteriormente y que han dado lugar a nuevas versiones del mismo.

4.1.1. Algoritmo ID3

ID3 es un algoritmo ideado por R. Quinlan en 1983 [185] utilizado para crear árboles de decisión clasificadores a partir de un conjunto de datos que debe estar conformado por una serie de *objetos* que son descritos a través de una colección de *propiedades*. Cada objeto del conjunto de entrenamiento pertenece a una *clase* (normalmente, representado por medio del valor del atributo *target*) de un conjunto de clases mutuamente excluyentes. ID3 ha sido una de las técnicas más utilizadas en aplicaciones de aprendizaje automático, y ha sido aplicado a tareas tan diversas como predicción de epidemias, control de robots o clasificación automática de clientes en bancos o entidades aseguradoras [225, 211, 105].

Este algoritmo realiza un proceso de inducción para construir una serie de reglas que permitan clasificar cada uno de los objetos en el conjunto de entrenamiento en la clase correspondiente. La primera pregunta a realizar para construir reglas de este tipo es: ¿qué propiedades de los objetos son adecuadas (proporcionan suficiente información) para realizar esta tarea? Hay condiciones a priori que podemos determinar para la viabilidad de esta labor. Por ejemplo, si el conjunto de entrenamiento contiene objetos con exactamente las mismas propiedades y que pertenecen a clases distintas, es claro que es imposible diferenciar los objetos teniendo en cuenta las propiedades disponibles. En este caso, los atributos tenidos en cuenta no serán los adecuados para realizar la tarea de inducción (o se ha podido producir un error en la toma de datos previa al análisis). En caso contrario, siempre es posible construir un árbol de decisión que permita clasificar correctamente cada objeto de nuestro conjunto, ya que si el conjunto de entrenamiento es finito y permite una clasificación correcta, siempre podemos construir un árbol que recorra todas las posibles combinaciones

de valores de los atributos construyendo un árbol con la anchura y profundidad adecuadas.

Sin embargo, normalmente no estamos interesados en un árbol cualquiera de decisión, buscamos un árbol simple que dé explicación a la clasificación, y para conseguir un árbol de tamaño menor podemos hacer uso del concepto de *reducción de impureza* (que será presentado en una sección posterior) para decidir qué propiedades de los objetos son las que nos permiten construir las mejores reglas clasificadoras en cada caso.

Algorithm 1 ID3($Examples, TargetAttribute, Attributes$) [33]

```
1: Create a Root node for the tree
2: if Stop criteria is reached then
3:    return The single-node tree Root, with most frequent label in Examples.
4: else
5:    A = The Attribute in Attributes that best classifies Examples.
6:    for each possible value, v_i, of A do
7:       Add a new tree branch below Root, corresponding to the test A = v_i.
8:       Let Examples(v_i) be the subset of examples that have the value v_i for
         A.
9:       if Examples(v_i) is empty then
10:         Below this new branch add a leaf node with label = most common
            target value in the examples
11:      else
12:         Below this new branch add the subtree
            ID3(Examples(v_i), TargetAttribute, Attributes \ {A})
13:      end if
14:   end for
15:   return Root
16: end if
```

El algoritmo comienza creando un árbol que contiene un único nodo (nodo raíz) y al que están asociados todos los objetos pertenecientes al conjunto de entrenamiento. En cada paso, el algoritmo evalúa qué propiedad permite dividir de mejor manera el conjunto de objetos actual (máxima reducción de impureza) y se elige para ser evaluada en el nodo actual. A continuación se crean tantas ramas como valores posibles toma dicha propiedad y se transmiten por ellas los objetos que tienen el valor asociado, además se elimina la propiedad utilizada del conjunto de posibles propiedades a utilizar de manera que no pueda ser utilizada

en los nodos que derivan del actual (tampoco serviría de nada mantenerla, ya que no aportará más información en las sucesivas agrupaciones conseguidas). En el caso de que un nodo herede un conjunto de objetos que pertenecen a una única clase, este nodo se convertirá en una hoja asociada a dicha clase y en él no se repite el proceso (la pureza del conjunto de objetos asociado ya es máxima). Este proceso se repite en los nodos que presentan alguna impureza de manera recursiva hasta que se alcanza algún criterio de parada (algunos de los criterios de parada disponibles serán presentados en la sección 4.1.3).

Debemos indicar que es conveniente que el número de ejemplos utilizados para la construcción del árbol de decisión sea muy superior al de posibles clases, ya que el proceso de generalización está basado en un análisis estadístico que no podría distinguir patrones de interés a partir de coincidencias aleatorias. Dependiendo del número de atributos a considerar y de la distribución de las muestras en el espacio de entrada, un problema real puede llegar a requerir cientos de miles de ejemplos de entrenamiento [135].

Podemos ver el proceso seguido por el algoritmo ID3 como una búsqueda en un espacio de árboles de decisión. El algoritmo realiza una *búsqueda en escalada* en este espacio comenzando con un árbol de decisión que posee sólo un nodo (el nodo raíz del resto de árboles) que contiene todas las muestras y, en pasos sucesivos, se van añadiendo hijos al árbol de forma recursiva disminuyendo el grado de impureza de las hojas.

Para un conjunto de atributos prefijado, el espacio de todos los árboles de decisión que se pueden construir es el *espacio completo* de funciones discretas finitas definidas en el producto de los espacios de valores de los atributos [158]. Además, la búsqueda realizada por ID3 sigue el principio de *divide y vencerás*, realizando una división recursiva del árbol en subárboles en los que se busca una mayor homogeneidad (pureza) en las clases existentes, de tal forma que el proceso se realiza hasta que cada partición contenga ejemplos que pertenecen a única clase (o más comúnmente, hasta que se alcance alguno de los criterios de parada).

Desde el punto de vista del proceso de búsqueda, la estrategia utilizada por este algoritmo pertenece a la categoría de los métodos que llevan a cabo una búsqueda *voraz* (se elige la mejor opción en cada paso y no se reconsideran las decisiones anteriores en pasos sucesivos, no hay, por tanto, ningún tipo de retroceso ni se consideran las consecuencias a largo plazo de las opciones no elegidas), y por ello tiene el problema de que puede converger a óptimos locales que no sean óptimos globales (es decir, son solución, posiblemente buena, pero no necesariamente la mejor).

La heurística que sigue ID3 consiste en preferir árboles que tengan atributos con mayor aporte de información (disminuyen más la impureza) más cerca de la

raíz, por lo que se suelen obtener árboles relativamente cortos. También debido a la heurística que emplea, favorece atributos con muchos valores.

En cuanto a la complejidad del algoritmo, el espacio ocupado por ID3 es el espacio ocupado por el árbol de decisión *actual*. En el peor de los casos, habrá un nodo hoja por cada ejemplo, con un número de nodos internos igual al número de hojas menos uno, que se puede producir cuando en el conjunto de entrada cada ejemplo pertenece a una clase diferente. En el mejor de los casos, el árbol poseería un único nodo, que se produciría cuando todos los ejemplos del conjunto de entrenamiento pertenecen a la misma clase. El tiempo de ejecución del algoritmo crece linealmente con el número de ejemplos de entrenamiento y exponencialmente con el número de atributos [144].

Pasemos ahora a analizar algunas de las formas más habituales de calcular la reducción de impureza, las diferentes condiciones de parada que se pueden considerar, las técnicas de poda que permiten que los árboles generados ofrezcan una mejora en rendimiento y expresividad, y algunas de las modificaciones que han permitido mejorar su eficiencia a lo largo de los años.

4.1.2. Medidas de Impureza

Hemos visto que en cada paso de la construcción del árbol de decisión se debe elegir el atributo que permita dividir mejor al conjunto de objetos, aunque no se ha especificado cómo calcular la reducción de impureza. Para ello, normalmente se utiliza el concepto de *ganancia de información* como medida, pero hay numerosos criterios para seleccionar el atributo que mejor divide a un conjunto determinado [203]. Veremos a continuación las medidas de impureza más comunes en la literatura, pero antes definamos formalmente qué entendemos por impureza.

Definición 35. *Dada una variable aleatoria X, con k posibles valores discretos, que sigue una distribución $P = (p_1, p_2, ..., p_k)$ (es decir, $P[X = x_i] = p_i$), una medida de impureza para X es una función $\phi : [0,1]^k \to [0,1]$ que satisface las siguientes condiciones:*

- $\phi(P) = 0$ *si existe* i $(1 \leq i \leq k)$ *tal que* $p_i = 1$.
- $\phi(P) = 1$ *si para todo* i $(1 \leq i \leq k)$ *se tiene que* $p_i = 1/k$.
- $\phi(P)$ *es simétrica con respecto a los elementos de* P.
- $\phi(P)$ *es continua y diferenciable.*

Obsérvese que se pueden identificar las distribuciones de probabilidad sobre X con elementos $(p1, \ldots, p_k) \in [0,1]^k$ tales que $\sum_i p_i = 1$.

Definición 36. *Dado un conjunto de entrenamiento S, formado por un conjunto de elementos compuestos por pares de la forma $\{(A:v)\}_{A \in \mathcal{A}}$, donde $\mathcal{A}$ es un conjunto de atributos, y notando $S_{A=v}$ como el subconjunto de elementos en S que contienen el par $(A:v)$ (es decir, poseen el valor v en su atributo A), podemos definir el* vector de probabilidad del atributo A *(que supondremos que puede tomar los valores $\{c_1, \ldots, c_k\}$) como:*

$$P_A(S) = (\frac{|S_{A=c_1}|}{|S|}, ..., \frac{|S_{A=c_k}|}{|S|})$$

Es decir, para calcular el vector de probabilidad $P_A(S)$ de un conjunto de datos S según su propiedad A sólo debemos calcular la proporción de objetos asociados a cada posible valor de A.

Como nuestro objetivo es usar árboles de decisión para aprender un clasificador (o un regresor) a partir de los datos definidos por medio de sus atributos, destacamos uno de los atributos, $Y \in \mathcal{A}$, como el atributo objetivo, o *target*. En este sentido, podemos medir la variación de impureza que se obtiene respecto a la clasificación (regresión) que se realiza de este atributo:

Definición 37. *Fijada una función de impureza, ϕ, la* bondad *de la división realizada por un atributo discreto se define como la reducción de impureza con respecto al atributo target, Y, después de particionar S de acuerdo a los valores v_j de A:*

$$\Delta\phi(A,S) = \phi(P_Y(S)) - \sum_{v_j=1}^{|rang(A)|} \frac{|S_{A=v_j}|}{|S|} \phi(P_Y(S_{A=v_j}))$$

Existen numerosas medidas de impureza que cumplen con las restricciones presentadas y que, por tanto, pueden ser utilizadas para medir la bondad de las divisiones en cada uno de los pasos del algoritmo ID3. Las más comunes son:

Ganancia de Información

La ganancia de información [186] es un criterio de división que utiliza la *entropía* como medida de impureza.

Dada una variable aleatoria discreta X, podríamos plantearnos cuánta información recibimos al conocer el valor de X en un determinado evento. Si

sabemos que un evento $X = x_i$ es muy improbable, estaremos recibiendo mucha información al descubrir que este evento ha ocurrido. Por el contrario, si tenemos la certeza de que un evento $X = x_i$ va a ocurrir seguro no recibiremos ninguna información al descubrirlo. La cantidad de información, $I(X = x_i)$, que recibimos al conocer el evento x_i puede entenderse como la *cantidad de sorpresa* recibida al descubrir ese valor de X, por lo que tiene sentido imponer que I sea una función monótona decreciente de la probabilidad p_i (cuanto mayor es la probabilidad del evento, menor es la información que proporciona).

Además, dados dos eventos, x e y, de la variable X no relacionados, la información obtenida debe ser la suma de las informaciones obtenidas al observar ambos eventos por separado, es decir:

$$I(x, y) = I(x) + I(y)$$

Sabemos que, desde el punto de vista de la distribución de probabilidad asociada, dos eventos no relacionados deben cumplir:

$$P(X = x, X = y) = P(X = x)P(X = y)$$

A partir de las dos ecuaciones anteriores, y usando las propiedades numéricas del logaritmo, tenemos la opción de definir la información a partir del logaritmo de p_i, de modo que podemos intentar definir:

$$I(x_i) = -\log p_i$$

Supongamos que un emisor quiere transmitir el valor de la variable aleatoria discreta X a un receptor. La información media que es transmitida en el proceso corresponde al valor esperado de la información con respecto a la distribución $p(X)$, y viene dada por:

$$\mathcal{E}(X) = -\sum_{i=1}^{|K|} p_i \log p_i$$

y que se denomina *entropía* de la variable aleatoria X. Se puede comprobar fácilmente que la función de entropía verifica las propiedades impuestas a las medidas de impureza anteriormente.

Debe indicarse que asignamos $p \log p = 0$ cuando se da el caso $p = 0$. La elección de la base para el logaritmo es libre aunque es común utilizar la base 2 debido a que de esta forma la entropía se refiere a la cantidad de bits promedio que se requieren para codificar el valor de la variable X [217].

En el caso que nos ocupa, la *Ganancia de Información* (Information Gain) que aporta la división del conjunto de datos respecto de un atributo $A \in \mathcal{A}$ puede ser definida como la reducción de entropía tras haber hecho la división respecto de ese atributo (la distribución de probabilidad asociada vendrá dada por la frecuencia asociada a la aparición de cada valor de los atributos):

$$IG(A,S) = \mathcal{E}(y,S) - \sum_{v_j \in rang(A)} \frac{|S_{A=v_j}|}{|S|} \mathcal{E}(y, S_{A=v_j})$$

donde:

$$\mathcal{E}(y,S) = \sum_{c_j \in rang(y)} -\frac{|S_{y=c_j}|}{|S|} log_2 \frac{|S_{y=c_j}|}{|S|}$$

Esta medida de impureza es con la que se presentó originalmente el algoritmo ID3 ideado por R. Quinlan. Aunque es utilizada normalmente como una medida de reducción de impureza adecuada para seleccionar qué atributo divide mejor a un conjunto de ejemplos dado, no es perfecta. Existe un problema notable cuando la ganancia de información se aplica a atributos que pueden tomar un valor de entre un gran número de posibilidades. Por ejemplo, supongamos que estamos construyendo un árbol de decisión para datos que describen el comportamiento de los clientes en un negocio. Si se tuviera en cuenta, por ejemplo, el número de tarjeta de crédito como un atributo de los clientes, dicho atributo aportaría una gran ganancia de información pues (normalmente) identifica unívocamente a cada cliente en el sistema, pero no es coherente utilizarlo como un atributo clasificador ya que provoca un claro sobreajuste en el aprendizaje. Para evitar este problema, se utiliza en ocasiones la ganancia de información normalizada que definimos a continuación.

Ganancia de Información Normalizada

La *Ganancia de Información Normalizada* (Gain Ratio) condiciona la construcción del árbol para que no considere atributos con un gran número de posibles valores. Aunque resuelve el problema apuntado en el parágrafo anterior, presenta la desventaja de que atributos con valores de aporte de información muy bajos pueden verse afectados negativemente. Quinlan [188] propuso la ganancia de información normalizada como una medida que *normaliza* la ganancia de información de la siguiente forma:

$$GR(A,S) = \frac{IG(A,S)}{\mathcal{E}(A,S)}$$

Nótese que esta medida no estará definida para el caso en el que el denominador sea nulo. Por lo que habrá que definir dicho caso como especial. Quinlan demostró que la Ganancia de Información Normalizada consigue mejores resultados tanto en precisión de resultados como en la complejidad de ejecución con respecto a la Ganancia de Información [184].

Impureza de Gini

La *Impureza de Gini* es una medida de impureza que calcula la divergencia entre distribuciones de probabilidad [41]. En concreto, en el caso de la inducción de árboles de decisión, calcula la divergencia entre diferentes distribuciones de probabilidad del atributo *target*.

Se define formalmente como:

$$Gini(A,S) = 1 - \sum_{c_j \in rang(A)} (\frac{|S_{A=c_j}|}{|S|})^2$$

Por lo que el criterio de evaluación para seleccionar el atributo A quedaría:

$$GiniGain(A,S) = Gini(Y,S) - \sum_{v_j \in rang(A)} \frac{|S_{A=v_j}|}{|S|} Gini(Y, S_{A=v_j})$$

El índice de Gini ha sido la medida de impureza utilizada en el algoritmo CART así como en otros trabajos relacionados con árboles de decisión [89].

4.1.3. Criterios de Parada

Como se ha comentado, la construcción del árbol de decisión a través del algoritmo ID3 continúa hasta que algún criterio de parada es alcanzado. Los criterios de parada más comunes son [203]:

1. Todas las instancias en el conjunto de entrenamiento del nodo pertenecen a un mismo valor del atributo *target*, Y.

2. La profundidad máxima del árbol ha sido alcanzada.

3. El nodo actual tiene un número de elementos asociados menor que un umbral determinado.

4. Ningún atributo disponible aporta menos reducción de impureza que un umbral determinado.

El primer criterio puede producir árboles sobreajustados a los datos de entrenamiento, mientras que los otros criterios pueden provocar que la construcción del árbol finalice antes de que se haya obtenido toda la *información útil* del conjunto de entrenamiento.

También se puede usar un criterio de parada basado en la división del conjunto de entrenamiento, $S = S_{train} \cup S_{test}$, que vimos en el capítulo 2, con el fin de controlar los efectos del sobreajuste.

4.1.4. Procesos de Poda

Utilizar un criterio de parada *estricto* favorece la creación de árboles de decisión pequeños y poco ajustados a los datos en el conjunto de entrenamiento (así que se espera que generalicen bien). Por otro lado, utilizar un criterio de parada *suave* tiende a generar árboles de decisión grandes y sobreajustados a los datos en el conjunto de entrenamiento.

Los métodos de poda sugeridos originalmente por Breiman et al. [41] fueron desarrollados para solucionar este dilema. La idea es permitir que los árboles crezcan hasta sobreajustarse a los datos en el conjunto de entrenamiento y una vez finalizada la construcción del árbol éste es *recortado* convirtiendo los subárboles que menos contribuyen a la reducción del error generalizado en hojas para obtener un árbol más pequeño y menos sobreajustado [203]. Estos procesos de recorte se conocen con el nombre genérico de *podas*.

La poda es muy útil cuando el objetivo del aprendizaje es construir modelos sencillos pero con una buena eficiencia en la predicción. Se ha demostrado en varios estudios que el empleo de técnicas de poda puede mejorar la tarea de generalización de los árboles de decisión, especialmente en dominios ruidosos. Otra de las claves del uso de técnicas de poda es obtener el equilibrio entre precisión en la clasificación y simplicidad en el modelo [40].

Existen varias técnicas para podar los árboles de decisión y la mayoría de ellas llevan a cabo un recorrido *top-down* o *bottom-up* a través de los nodos del árbol, cada nodo será podado si su eliminación mejora ciertos criterios en el árbol. Las técnicas de poda más populares son: *poda del enlace más débil* [41], *poda por reducción de error* [186], *poda pesimista* [188], y *poda basada en el error* [203], introducidas en mejoras sucesivas del algoritmo ID3.

4.1.5. Algoritmo C4.5 y mejoras adicionales

EL algoritmo C4.5 es una mejora del algoritmo ID3 presentada también por Quinlan [188] que cubre algunas deficiencias del ID3 original y presenta algunas diferencias notables: utiliza la Ganancia de Información Normalizada como criterio de división del conjunto de objetos, permite trabajar con atributos continuos, la división termina cuando el número de objetos en una rama es menor que un umbral determinado, y utiliza poda basada en el error (EBP).

Además, C4.5 permite trabajar con un conjunto de objetos con *valores faltantes*, estos valores simplemente no se usan en los cálculos de la ganancia y la entropía cuando intervienen los atributos con valores desconocidos [60, 203].

Para el tratamiento de valores continuos en sus atributos este algoritmo, esencialmente, efectúa un proceso de clusterización (agrupamiento) y crea artificialmente nuevos atributos que representan a los diversos clusters (agrupaciones) que ha encontrado.

Como es muy común que los atributos continuos sean valores de un espacio continuo completamente ordenado (como $\mathbb{R}$) en su diseño hace uso de un mecanismo de clusterización restringido pero que es especialmente eficiente desde el punto de vista computacional. Para ello, se ordenan los valores existentes para el atributo y se especifica la clase a la que pertenecen los ejemplos que corresponden a cada uno de dichos valores, eliminando los demás atributos. Se agrupan estas clases y se consideran puntos intermedios en los que se producen los diversos cambios de clase observados.

Posteriormente, S. Ruggieri mejoró el algoritmo C4.5 generando el EC4.5 (C4.5 eficiente) [207]. Los resultados muestran que para un mismo árbol de decisión la eficiencia de EC4.5 es seis veces mayor que la de C4.5, pero utiliza más memoria [142]. En 2003, C. Olaru desarrolló un árbol de decisión basado en lógica difusa llamado *Soft Decision Tree* [173] que sintetiza la generación y la poda de árboles de decisión a la hora de construirse, e incorpora determinadas técnicas para mejorar la capacidad de inducción del árbol. La exactitud del árbol de decisión difuso es mayor que la del árbol de decisión general.

Además, se han planteado versiones incrementales del algoritmo como son ID4 [135] e ID5 [234]. En el caso de ID4, el algoritmo mantiene estadísticas de la distribución de los valores de los atributos clasificados debajo de cada nodo y si el valor de reducción de impureza de un atributo baja con respecto al de otro, el subárbol se rehace. Desde el punto de vista de búsqueda tiene un operador de especialización (crecer un árbol) y un operador de generalización (borrar un subárbol), lo que realmente supone un retroceso simulado. En el caso del ID5 en lugar de borrar o hacer crecer subárboles, los reorganiza si determina que se han degradado. Generalmente, necesita muchos menos ejemplos de entrenamiento

que el algoritmo original, aunque es computacionalmente más costoso.

4.1.6. Random Forest

De forma genérica, se denominan *Random Forest* (Bosques Aleatorios) a los métodos agregados que hacen uso de árboles de decisión como elementos constitutivos básicos.

Todos los métodos de combinación que vimos en los Fundamentos se pueden aplicar sobre árboles de decisión para conseguir Random Forest de manera eficiente, aunque el más común suele ser el de los subespacios aleatorios.

En la literatura se pueden encontrar numerosas referencias que describen al detalle este tipo de métodos agregados [141, 235], los cuales resultan especialmente interesantes a la hora de extender las capacidades de predicción de árboles aislados y de manera independiente al tipo de test que se realice en los nodos internos del árbol, verificación de atributos, predicados lógicos o evaluación multi-relacional.

4.2. Árboles de Decisión Multi-Relacionales

Hasta ahora hemos realizado un análisis de diferentes técnicas de generación automática de árboles de decisión a partir de datos *planos*, es decir, datos en forma de tabla en la que cada fila representa un objeto y cada columna alguna de sus propiedades.

Uno de los objetivos que se persiguen en esta tesis es ofrecer metodologías que permitan llevar a cabo tareas de aprendizaje automático a partir de datos en forma de grafos con propiedades, por lo que el número de posibles *atributos* de cada elemento va mucho más allá de las propiedades que tiene asociadas directamente por medio de la función μ. En un grafo con propiedades, las propiedades de los elementos que se relacionan con un objeto determinado también pueden ser considerados atributos del mismo, es más, incluso la estructura topológica que forman los elementos en su *entorno* y las diversas medidas que se pueden tomar de esa estructura podrían ser consideradas como atributos.

A continuación vamos a presentar las técnicas de aprendizaje automático y minería de datos relacionales más importantes que podemos encontrar en la literatura, ya sea por su capacidad de predicción, su eficiencia computacional, o porque han servido de base a otras técnicas relevantes.

4.2.1. Pogramación Lógica Inductiva

La Programación Lógica Inductiva (ILP) [179] es un área del aprendizaje automático que utiliza fundamentos de Programación Lógica para representar de manera uniforme ejemplos, base de conocimientos, e hipótesis.

Dada una codificación de la base de conocimientos y un conjunto de ejemplos representados a través de una base de datos lógica de hechos, un sistema ILP obtendrá un programa que discriminará todos los ejemplos positivos de los negativos. La potencia de ILP es su expresividad, ya que las reglas que extrae son interpretables por un humano, pero su desventaja suele radicar en su ineficiencia a la hora de trabajar con bases de datos complejas, así como su incapacidad para trabajar adecuadamente con atributos continuos [176].

A pesar de que la ILP por sí misma (sin una transformación adecuada de las relaciones entre datos a predicados lógicos) no nos va a permitir generar árboles de decisión relacionales, sí nos permite generar de manera automática árboles de decisión lógicos que pueden ser considerados la base de uno de los algoritmos más importantes de generación automática de árboles de decisión relacionales y que será presentado más adelante (hablamos, de nuevo, de MRDTL).

En el paradigma de aprendizaje a partir de interpretaciones de la Programación Lógica Inductiva [189] cada ejemplo es un conjunto de cláusulas que codifican sus propiedades (una base de datos de conocimiento en Prolog [55], por ejemplo), y está clasificado en una de entre las posibles clases. Además, se debe especificar una base de conocimientos expresada también a través de un conjunto de cláusulas. Formalmente, podemos expresar el problema a resolver de la siguiente forma:

Definición 38. *Dado un conjunto de clases, C, un conjunto de ejemplos clasificados, E, y una base de conocimientos, B, el problema consiste en encontrar una hipótesis H (un conjunto de cláusulas) tal que para cada ejemplo, $e \in E$, se tiene:*

- $(H \wedge e \wedge B) \models c$,
- $(H \wedge e \wedge B) \not\models c'$

donde c es la clase a la que pertenece el ejemplo, y $c' \in C - \{c\}$.

A continuación presentamos los árboles de decisión lógica, así como un algoritmo para la generación automática de los mismos, a partir de un conjunto de ejemplos previamente clasificados, una base de conocimientos, y un lenguaje que indica qué tipo de preguntas están permitidas en el árbol.

Árboles de Decisión Lógica

Un *Árbol de Decisión Lógica* es un árbol de decisión binario que verifica [32]:

- Todos los test de los nodos internos se expresan como conjunción de literales (en una Lógica de Primer Orden prefijada).
- Una variable introducida en un nodo determinado del árbol (no aparece en sus ancestros) no puede aparecer en su subárbol derecho.

Un ejemplo de árbol de decisión lógica se muestra en la Figura 4.1, que clasifica un objeto en una de las tres posibles clases (*sendback*, fix, *ok*) a través de una serie de conjunciones de predicados aplicados sobre el mismo [33].

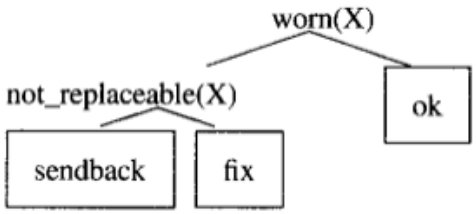


Figura 4.1: Árbol de decisión lógica.

Se utilizará la siguiente notación: un nodo es, o bien una hoja perteneciente a la clase k ($leaf(k)$), o bien un nodo interno con una conjunción c asociada, donde su rama izquierda será denotada por l, y la rama derecha por r ($node(c, l, r)$) [33].

El Algoritmo 2 implementa un árbol de decisión lógica para clasificación evaluando si un ejemplo, e, es consistente con una clase, c, en el contexto de una base de conocimientos, B. El conjunto de literales, C, va acumulando la información otorgada por cada nodo en el camino desde la raíz hasta la hoja, de esta forma se verifica que cumple con todas las conjunciones impuestas. Nótese que cuando se clasifica el objeto a la derecha, C no necesita ser actualizado, una evaluación fallida no introduce nuevas variables [33].

El Algoritmo 3 se corresponde con el algoritmo TILDE (Top-Down Induction of Logical Decision Trees) [32] y construye árboles de decisión lógica (como el utilizado por el Algoritmo 2 para clasificar instancias) a partir de un conjunto de ejemplos clasificados, E, una base de conocimientos, B, y un lenguaje, Q, que indica qué tipo de preguntas están permitidas en el árbol [33].

Algorithm 2 Classify_ILP(e) [33]

$C := true$
$N := root$
while $N \neq leaf(c)$ **do**
  $N = node(conj, left, right)$
  if $C \wedge conj$ succeeds in $B \wedge e$ **then**
    $C := C \wedge conj$
    $N := left$
  else
    $N := right$
  end if
end while
return c

Algorithm 3 Build_Logic_Tree(E, B, Q) [33]

Create a *Root* node for the tree
if Stop criteria is reached **then**
  return The single-node tree *Root*, with most frequent label in E.
else
  Q_b = The element in Q that best classifies E.
  $Q\prime := Q \setminus Q_b$
  $E_1 := \{e \in E | e \cup B \models Q_b\}$
  $E_2 := \{e \in E | e \cup B \not\models Q_b\}$
  return node(Q_b, Build_Logic_Tree(E_1,B,$Q\prime$), Build_Logic_Tree(E_2 ,B, $Q\prime$))
end if
return *Root*

El único punto en el que este algoritmo se diferencia del ID3 presentado por Quinlan (sin tener en cuenta las posibles optimizaciones implementadas en C4.5 u otros) es con respecto a los tests llevados a cabo en cada nodo del árbol.

Tras el auge de la ILP se lograron algunos avances importantes en la minería de datos multi-relacional [122, 256]. Yin Xiaoxin [252] diseñó *CrossMine*, un modelo de clasificación multi-relacional que mezcla ILP y las bases de datos relacionales, mejorando la eficiencia en este tipo de tareas a través de un método para realizar uniones virtuales de tablas en bases de datos relacionales [140].

4.2.2. Multi-Relational Decision Tree Learning (MRDTL)

MRDTL es un algoritmo para el aprendizaje de árboles de decisión multi-relacionales [139] basado en las ideas de Knobles et al. [129], que trabaja con el concepto de Grafo de Selección (Selection Graph), presentado en el capítulo 3. Cabe destacar que la especificación de este método está orientada a bases de datos relacionales debido, en parte, a que en el tiempo en el que se presentó aún no se habían desarrollado otro tipo de bases de datos más adecuadas para este tipo de tareas.

Esencialmente, MRDTL, como el resto de algoritmos que generan árboles de decisión clasificadores a partir de un conjunto de ejemplos, añade nodos de decisión al árbol a través de un proceso de refinamiento recurrente hasta que se verifica algún criterio de parada. Cuando esto ocurre, se introduce un nodo hoja con la correcta clasificación.

La gran diferencia que introduce este método con respecto a los presentados anteriormente es que utiliza grafos de selección como atributos binarios en cada nodo de decisión del árbol, que tendrá dos ramas que lo conectarán con sus nodos hijos: la rama positiva (a la que se transmitirán los objetos/registros que cumplan con el grafo de selección asociado) y la rama negativa (a la que se transmitirán los objetos/registros que no cumplan con el grafo de selección asociado). La elección del nodo de decisión a añadir en cada paso está guiada por alguna medida de impureza.

MRDTL comienza con un único nodo (la raíz del árbol), que representa a todos los registros en el conjunto de entrenamiento y que tendrá asociado un grafo de selección compuesto a su vez por un único nodo que se corresponde con la tabla *target* (tabla del modelo relacional donde se encuentran los registros a clasificar). El pseudocódigo del algoritmo MRDTL se presenta en el Algoritmo 4 [129].

La función $Optimal\_Refinement$ considera cada posible refinamiento que puede hacerse al grafo de selección (patrón) actual, Q, con respecto a la base

Algorithm 4 $MRDTL(E$: *registers to classify*, D : *database*, Q : *selection graph*)

1: Create a *Root* node for the tree
2: **if** Stop criteria is reached **then**
3: **return** The single-node tree *Root*, with most frequent label in E.
4: **else**
5: $R = Optimal\_Refinement(E, Q, D)$
6: $Q' = R(Q)$
7: $\bar{Q}' = \bar{R}(Q)$
8: $E_1 = \{e \in E | Q'(e)\}$
9: $E_2 = \{e \in E | \bar{Q}'(e)\}$
10: Add 2 new tree branches below *Root* with values $MRDTL(E_1, D, R(Q))$ and $MRDTL(E_2, D, \bar{R}(Q))$.
11: **return** *Root*
12: **end if**

de datos, D, y selecciona, de manera voraz, el refinamiento óptimo (aquel que minimice la medida de impureza seleccionada). El conjunto de posibles refinamientos que se pueden aplicar en cada caso estará gobernado por el grafo de selección actual y por la estructura de la base de datos, D. $\bar{R}$ denota el refinamiento complementario a R (el concepto de refinamiento complementario se explica a continuación).

Refinamientos

La forma de ir refinando el grafo de selección asociado a cada uno de los nodos internos del árbol de decisión es a través de un conjunto de posibles refinamientos que permiten definir hipótesis que sean consistentes con los datos. A continuación presentamos los refinamientos introducidos en [129] y que serán ilustrados utilizando el grafo de selección de la figura 4.2.

Los algoritmos que inducen árboles de decisión requieren que los subconjuntos pertenecientes a patrones derivados del mismo nodo padre sean mutuamente excluyentes. Por este motivo, los dos refinamientos (añadir condición a nodo, y añadir arista y nodo) son introducidos con su refinamiento complementario.

Nótese que el algoritmo MRDTL no permite trabajar con Grafos de Selección que presenten ciclos, por lo que a través de los refinamientos presentados sólo se podrán construir Grafos de Selección de tipo árbol.

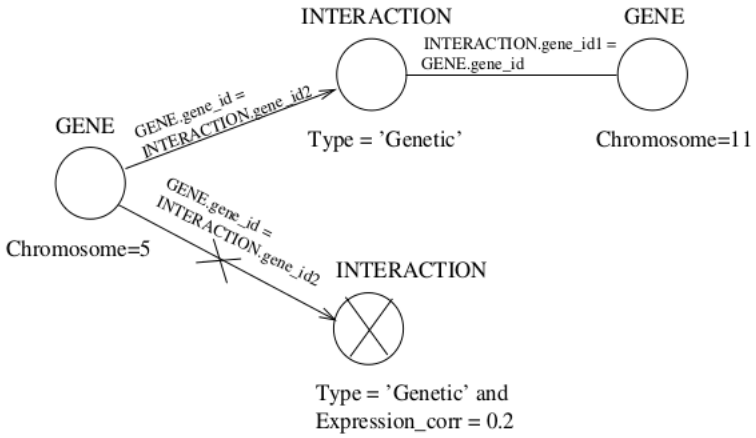


Figura 4.2: Grafo de selección.

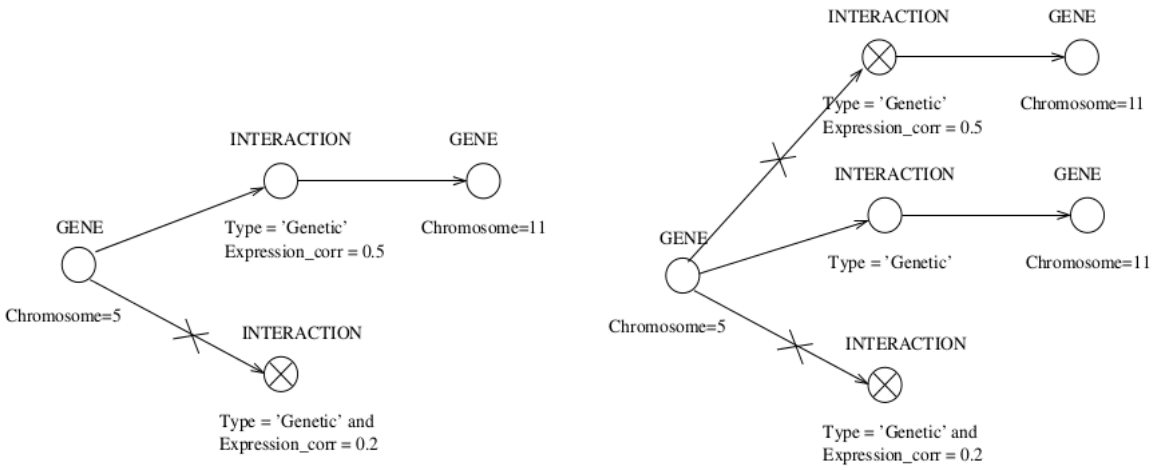
1. *Añadir condición*: Este refinamiento añade una condición al conjunto de condiciones asociado a alguno de los nodos sin modificar la estructura del grafo de selección. Para el grafo de selección de la Figura 4.2 añadir la condición `Expression_corr=0.5` al nodo `ITERACTION` produce el grafo de selección mostrado en la Figura 4.3(a).

2. *Añadir condición (negada)*: Este es el refinamiento complementario al anterior y se define como sigue: Si el nodo que está siendo refinado no representa a la tabla *target*, este refinamiento introduce una arista ausente desde el padre del nodo de selección a refinar a un nuevo nodo cerrado. Las condiciones del nodo a refinar y los nodos descendientes así como sus relaciones deben ser copiadas al nuevo nodo. Además, la nueva lista de condiciones será extendida añadiendo la condicion negada.

   Si el nodo que está siendo refinado representa a la tabla *target*, simplemente se le añade la condición negada. El grafo de selección resultante de añadir la misma condición que en el ejemplo anterior pero negada se muestra en la Figura 4.3(b).

3. *Añadir arista y nodo*: Este refinamiento introduce una arista presente y su correspondiente nodo abierto. Para el grafo de selección de la Figura 4.2 añadir una arista desde el nodo `GENE` a un nodo `COMPOSITION` produce el grafo de selección de la Figura 4.4(a).

4. *Añadir arista y nodo (negada)*: Este refinamiento es el complementario del anterior e introduce una arista ausente y un nodo cerrado. No tiene sentido que un nodo cerrado sea refinado añadiendo más condiciones a sus



(a) Añadir condición. (b) Añadir condición (negada).

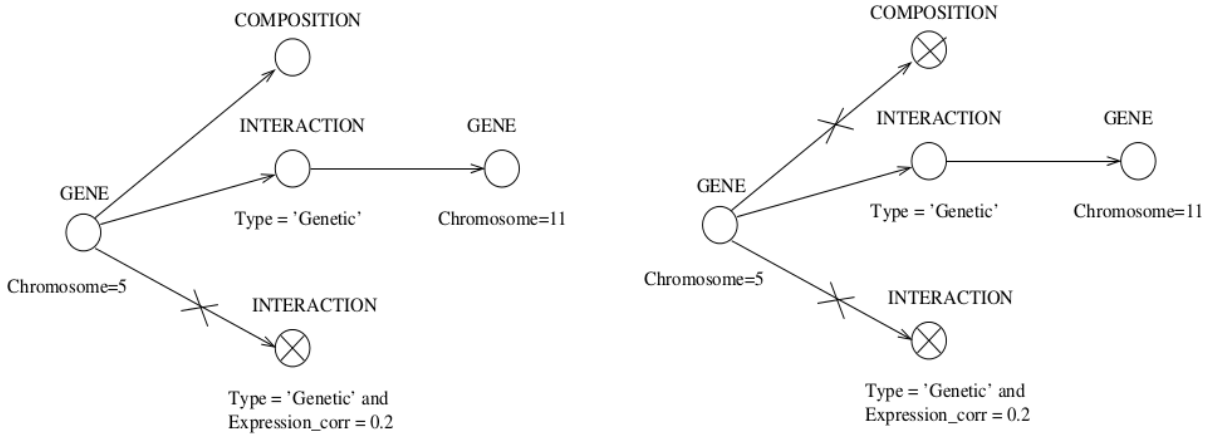
Figura 4.3: Refinamiento añadir condición (Grafos de selección).

atributos o a sus aristas salientes, debido a que es utilizado únicamente como herramienta auxiliar. Para el grafo de selección de la Figura 4.2, el refinamiento complementario de añadir una arista desde el nodo `GENE` a un nodo `COMPOSITION` produce el grafo de selección de la Figura 4.4(b).

El algoritmo MRDTL, los grafos de selección y sus refinamientos, suponen una primera aproximación a la inducción de árboles de decisión multi-relacionales pero poseen grandes invonvenientes en cuanto a su eficiencia y potencia. Además de que sólo pueden evaluar registros (no estructuras más complejas), los patrones aprendidos no pueden contener ciclos y sólo representan predicados lógicos sobre el registro bajo evaluación. Estos predicados sólo permiten evaluar la participación del registro actual en caminos de determinado tipo, no permiten realizar evaluaciones completas sobre estructuras en las que participa y con las que está relacionado.

Los experimentos realizados con MRDTL demostraron rendimientos comparables a los obtenidos con otros algoritmos de aprendizaje de la época sobre datos multi-relacionales. Sin embargo, MRDTL tiene dos limitaciones importantes desde el punto de vista del aprendizaje de datos multi-relacionales en grandes conjuntos de datos:

- Tiempo de ejecucion lento: Los experimentos de evaluación de rendimiento realizados con MRDTL en los datos de la KDD Cup 2001 [50] muestran que la ejecución de las consultas SQL derivadas de los grafos de selección



(a) Añadir arista y nodo.

(b) Añadir arista y nodo (negada).

Figura 4.4: Refinamiento añadir arista y nodo (Grafos de selección).

suponen un gran cuello de botella en términos del tiempo de ejecución del algoritmo.

- Imposibilidad de trabajar con *valores faltantes* en los atributos: En las bases de datos multi-relacionales es habitual encontrar datos a los que le falta algún valor en los atributos. Aunque ha habido implementaciones del algoritmo MRDTL [139] que han intentado tratar con ellos, lo hacen como un caso especial y no incluyen ninguna técnica estadística bien fundada para tratar este problema. Por tanto, la precisión de los árboles de decisión construidos están lejos de lo óptimo en la tarea de clasificación cuando se encuentran numerosos valores faltantes [256].

El algoritmo MRDTL-2 [22] nace con el objeto de resolver las limitaciones mostradas en el original, y presenta algunas mejoras:

- MRDTL-2 incluye técnicas para acelerar de manera significativa la ejecución de algoritmos de minería de datos multi-relacionales que, como MRDTL, se basan en el uso de Grafos de Selección, aplicando técnicas de optimización a las consultas SQL que derivan de la evaluación de los mismos.
- MRDTL-2 incluye una técnica simple y computacionalmente eficiente que utiliza clasificadores de Bayes para el completado de valores faltantes en atributos.

Además, para acelerar la ejecución, y debido a que determinadas conexiones no representan enlaces con una carga semántica considerable [140], MRDTL-2 sólo considera algunas de las conexiones presentes en la base de datos.

A pesar de estas mejoras, sigue presentando las limitaciones anotadas acerca de la capacidad de reconocer estructuras topológicas más complejas.

Ya vimos que nuestra propuesta de consultas en grafos, Property Query Graph, supone un avance con respecto a estas limitaciones y, en consecuencia, el algoritmo derivado de usar los PQG en la construcción de árboles de decisión multi-relacionaes puede suponer una ventaja sobre este método.

4.2.3. Decision Tree Graph-Based Induction (DT-GBI)

Graph-Based Induction (GBI) es una técnica de minería de datos que extrae patrones frecuentes (*network motifs*) de grafos etiquetados y dirigidos a través de la unión de pares de nodos conectados (pairwise chunking) [165] y que es muy eficiente debido a que utiliza una técnica voraz.

A partir de esta técnica, *Decisión Tree Graph-Based Induction* (DT-GBI) es un algoritmo de construcción de árboles de decisión para clasificar grafos utilizando los principios de GBI. En DT-GBI los atributos (llamados patrones o subestructuras) son generados durante la ejecución del algoritmo [87], por lo que DT-GBI es un generador de árboles de decisión con capacidad de construcción de atributos [164].

GBI ha sido utilizado para extraer patrones típicos en grafos dirigidos etiquetados, y ha demostrado su eficacia para resolver una gran variedad de problemas de aprendizaje mapeando las estructuras de los diferentes problemas a grafos dirigidos y etiquetados [254].

Comenzaremos describiendo el método GBI, así como algunas de sus mejoras, que servirá de base para poder presentar el algoritmo de construcción automática de árboles de decisión (DT-GBI).

Graph-Based Induction

En la definición original de GBI se asume que los patrones típicos en grafos representan *conceptos* y que su *tipicidad* es caracterizada por la frecuencia de aparición de dicho patrón o por el valor de alguna función de evaluación de su frecuencia [88]. Vamos a describir su funcionamiento a través de un ejemplo.

La Figura 4.5[88] muestra un patrón (sombreado) consistente en los nodos etiquetados con 1, 2 y 3 que es típico porque ocurre tres veces en el grafo.

GBI encuentra primero el par $1 \rightarrow 3$ basado en su frecuencia, une dichos nodos formando un nuevo nodo (nodo 10), en la siguiente iteración descubre el par $2 \rightarrow 10$ y lo une en un nuevo nodo (nodo 11), que representa el patrón sombreado.

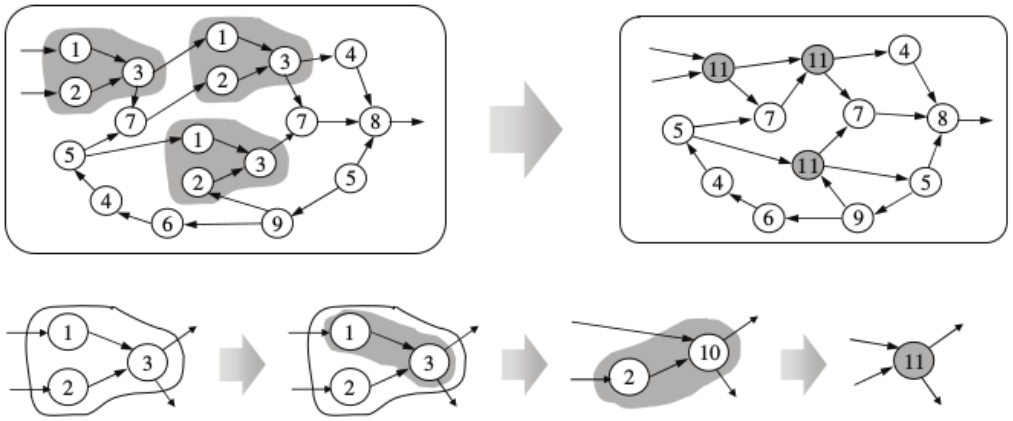


Figura 4.5: Método GBI.

Algorithm 5 GBI(G)

Require: G

1: Enumerate all the pairs P all in G
2: Select a subset P of pairs from P all (all the pairs in G) based on typicality criterion
3: Select a pair from P all based on chunking criterion
4: Chunk the selected pair into one node c
5: G_c := contracted graph of G
6: **while** termination condition not reached **do**
7: $P := P \cup GBI(G_c)$
8: **end while**
9: **return** P

Es posible extraer patrones típicos de varios tamaños repitiendo el procedimiento anterior, pero obsérvese que la búsqueda es voraz y no se aplica ningún procedimiento de *backtracking*. Debido a esto, no se garantiza que todos los *patrones típicos* que existen en el grafo de entrada sean extraídos. Además, GBI trata de extraer sólo los patrones típicos de un cierto tamaño. En GBI, para encontrar un patrón de interés todos los subpatrones presentes en él deben ser

también interesantes para que hayan sido seleccionados en algún momento. Esta limitación también está presente en el método MRDTL, y es parte inherente al método constructivo voraz que caracteriza los algoritmos derivados de ID3. Sin embargo, en algunos algoritmos es más evitable que en otros.

La salida del algoritmo GBI es un conjunto de patrones típicos ponderados. Es capaz de restaurar los patrones originales ya que se mantiene la información sobre qué nodo en la unión de un par es conectado durante el proceso recursivo [88].

Debido a que la búsqueda en el método GBI es voraz, los patrones que se extraen finalmente de un grafo a través de dicho método dependen de qué pares se hayan elegido en cada paso para su unión, por lo que puede haber determinados patrones *importantes* que no sean extraídos a través del método GBI. Para tratar de relajar este inconveniente se incorpora la técnica beam-search a GBI [145] incrementando el espacio de búsqueda para extraer patrones más representativos manteniendo la complejidad computacional a un nivel tolerable.

Decision Tree Graph-Based Induction

Para obtener DT-GBI a partir de GBI se consideran pares de la forma $(atributo : valor)$ para los grafos [87], donde *atributo* es un patrón o subgrafo, y *valor* indica la existencia, o no, del patrón en el grafo. Como los valores son binarios, los árboles de decisión construidos a través de este método también serán binarios.

Al construir un árbol de decisión, se enumeran todos los pares de nodos presentes en los datos y se selecciona un par. Los datos (grafos) son divididos en dos grupos: uno que contendrá los grafos que contengan el par seleccionado, y otro en el que estarán los grafos que no lo contengan. El par seleccionado es entonces colapsado (por medio de un nuevo nodo) en los grafos que lo contienen. Este proceso se aplica de manera recursiva en cada nodo del árbol de decisión, creando los atributos (pares de nodos) durante la ejecución. El algoritmo 6 muestra el pseudocódigo correspondiente, y la Figura 4.6 un ejemplo de árbol obtenido.

Este procedimiento hace que los atributos interesantes (patrones más complejos) para la tarea de clasificación se obtengan de forma constructiva. Por este motivo, DT-GBI puede ser interpretado como un método de extracción de características que son útiles para discriminar el conjunto de datos (grafos) [87].

Cabe destacar que el método DT-GBI construye árboles de decisión para clasificar grafos completos, a diferencia del método MRDTL y de nuestra propuesta, cuya misión es construir árboles de decisión que clasifiquen elementos inmersos de un grafo (nodos en el caso de MRDTL, y cualquier subestructura

Algorithm 6 *DT–GBI*(D) [88]

Create a *Root* node for the tree
if Stop criteria is reached **then**
  return The single-node tree *Root*, with most frequent label in D.
else
  $P = GBI(D)$
  Select a pair p from P
  Divide D into D_y (with p) and D_n (without p)
  Chunk the pair p into one node c
  D_{yc} = contracted data of D_y
  $DT_1 = DT\text{–}GBI(D_{yc})$
  $DT_2 = DT\text{–}GBI(D_n)$
  Augment *Root* by attaching DT_1 (DT_2) as its child along yes(no) branch
end if
return *Root*

en el caso de nuestra propuesta).

4.2.4. Random Forest Relacionales

De forma similar a como se definen Random Forest que hacen uso de árboles de decisión no relacionales, podemos considerar métodos agregados para mejorar la eficacia de los árboles de decisión relacionales y multi-relacionales que hemos visto en esta sección.

En [30] se presenta un método para la construcción de Random Forest Relacionales que asume independencia, en cuanto a la tarea de clasificación, entre las tablas de una base de datos, de tal manera que las predicciones basadas en árboles de decisión, que se han aprendido de diferentes tablas en la base de datos, se pueden combinar de manera probabilística. Para ello, se asignan diferentes pesos a cada uno de los árboles construidos a partir de las diferentes tablas y se eliminan los irrelevantes. Para la asignación de pesos a los árboles se hace uso de regresión logística. Este método, que ha sido evaluado en bases de datos del mundo real, ha demostrado buenos resultados.

En [235] hacen uso de Random Forest en el aprendizaje de datos relacionales trabajando con el algoritmo TILDE [33] como algoritmo base y utilizando consultas lógicas que pueden evaluar *resultados agregados* calculados a partir

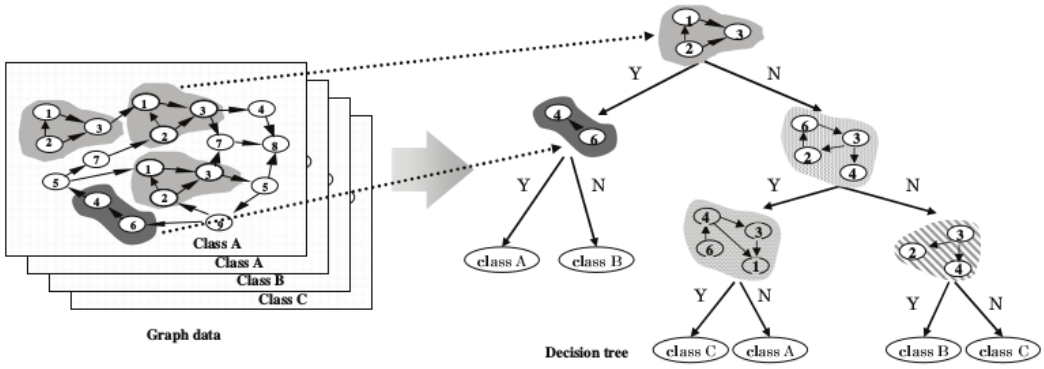


Figura 4.6: Método DT-GBI.

de las relaciones en la base de datos. Por ser un caso en el que el espacio de parámetros es muy elevado, Random Forest es un buen candidato para llevar a cabo el aprendizaje.

4.3. ID3 basado en Property Query Graphs

A continuación presentaremos PQG-ID3, una adaptación del algoritmo ID3 para crear árboles de decisión capaces de clasificar correctamente estructuras inmersas en un Grafo con Propiedades haciendo uso de Property Query Graphs como herramientas de test en cada nodo interno del árbol de decisión.

De forma similar a como trabaja MRDTL, nuestra propuesta buscará los PQG que mejor clasifiquen los ejemplos del conjunto de entrenamiento obteniendo los atributos a evaluar a lo largo de la construcción del árbol (extracción de características). Para generar un árbol que es capaz de obtener PQG interesantes a partir de uno inicial necesitamos utilizar refinamientos que den lugar a particiones encajadas del conjunto de estructuras bajo evaluación. De esta forma, en cada nodo interno del árbol se realizará una discriminación entre las estructuras que cumplen cada PQG resultante de un refinamiento. La forma en la que es elegido el mejor refinamiento en cada nodo del árbol de decisión será llevada a cabo haciendo uso de alguna de las medidas de impureza presentadas en la sección 4.1.2. De esta forma, la estrategia que elige la mejor ampliación del patrón asociado a un nodo interno es un hiper-parámetro del algoritmo.

El algoritmo PQG-ID3 posee una estructura muy similar a los diferentes algoritmos de construcción de árboles de decisión presentados a lo largo de este

capítulo, pero presenta la novedad de recibir como conjunto de entrenamiento estructuras (subgrafos) inmersas en un grafo con propiedades. Por medio de PQG adecuados, el árbol resultante de la ejecución del algoritmo podrá no solo evaluar propiedades de la estructura a clasificar, sino también propiedades de cualquier elemento (subgrafo) en su *entorno* (el entorno de una estructura puede llegar a contener todo el grafo en el que se encuentra inmersa).

El conjunto de entrenamiento completo, $\mathcal{L}$, estará formado por pares de la forma $(S, valor)$, donde S es un subgrafo de un grafo con propiedades $G = (V, E, \mu)$. En consecuencia, cada nodo, n, del árbol de decisión construido tendrá asociadas las siguientes estructuras:

- $\mathcal{L}_n = \{(S_1, y_1), ..., (S_N, y_N)\} \subseteq \mathcal{L}$, un subconjunto del conjunto de entrenamiento.
- $Q_n = (V_{Q_n}, E_{Q_n}, \alpha_{Q_n}, \theta_{Q_n})$, un PQG que verifican todos los subgrafos de $\mathcal{L}_n$.

El algoritmo 7 presenta formalmente el algoritmo PQG-ID3. Nótese que el conjunto de refinamientos disponibles, $REFS$, para ampliar el PQG en cada paso, permanece como un parámetro libre del modelo. La condición de parada y el criterio de selección del refinamiento en cada paso quedan abiertas en esta especificación del algoritmo.

Algorithm 7 PQG-ID3$(G, Q, \mathcal{L}, REFS)$

1: Create a *Root* node for the tree
2: **if** Stop criteria is reached **then**
3: **return** The single-node tree *Root*, with most frequent label in $\mathcal{L}$.
4: **else**
5: $(Q_1, .., Q_k) = Optimal\_Refinement(G, Q, L, REFS)$
6: $\mathcal{L}_1 = \{(S, y) \in \mathcal{L} : S \vDash Q_1\}, ..., \mathcal{L}_k = \{(S, y) \in \mathcal{L} : S \vDash Q_k\}$
7: Add k new tree branches below *Root* with values PQG-ID3$(G, Q_i, \mathcal{L}_i, REFS)$ for every $1 \leq i \leq k$.
8: **end if**

Obsérvese que la construcción del árbol se hace de forma recursiva, siendo la instrucción 3 la que representa el caso base que construye los nodos hojas. En un proceso habitual de aprendizaje a partir de subgrafos de G, los parámetros de la llamada inicial del algoritmo serán:

- $G = (V, E, \mu)$, grafo en el que se encuentran inmersos las estructuras a clasificar.

- $\mathcal{L} = \{(S_1, y_1), ..., (S_N, y_N)\}$, conjunto de pares (S_i, y_i) donde $S_i \subseteq G$ representa un subgrafo e y_i es su valor de salida asociado.
- $Q = (V_Q, E_Q, \mu_Q)$, PQG inicial (normalmente un PQG con la estructura común más grande en $S_1, ..., S_N$).
- $REFS$, conjunto de refinamientos disponibles.

El algoritmo sigue el procedimiento habitual en un algoritmo que sigue las directrices de ID3. Comienza creando un árbol que contiene un único nodo (nodo raíz) al que están asociados todos los objetos del conjunto de entrenamiento y cuyo PQG es el PQG inicial. Al nodo del árbol de decisión con el que estemos trabajando lo denominaremos *nodo actual* y al conjunto de ejemplos (pares) de entrenamiento asociado, *conjunto actual*. En cada paso, el algoritmo evalúa qué refinamiento permite dividir de mejor manera el conjunto actual (máxima reducción de impureza) y éste será elegido para ser aplicado al PQG inicial, y por tanto para discriminar los pares en el nodo actual. A continuación se crearán tantas ramas a partir del nodo actual como PQGs tenga el refinamiento elegido y por cada una de ellas se transmitirán los pares del conjunto actual que cumplan con el PQG asociado. Cada nodo hijo del nodo actual heredará cada PQG resultante del refinamiento y procederá a buscar (si no se alcanza la condición de parada) el mejor refinamiento para el nuevo PQG. De esta manera, por cada rama del árbol se heredará, no sólo un conjunto de pares, sino un PQG que verifican todos ellos. En caso de que se cumpla la condición de parada, el nodo se convertirá en una hoja asociada a la clase correspondiente. Este proceso se repite de manera recursiva.

Los árboles de decisión que se obtienen en la mayoría de los procedimientos automáticos dividen los datos en subconjuntos complementarios de manera recursiva [188]. Habitualmente, la división se lleva a cabo mediante la evaluación de una condición sobre el conjunto de objetos actual y su correspondiente división binaria: una rama recibirá el conjunto de objetos que cumplen con la condición, mientras que la otra recibe aquellos que no la cumplen. En un entorno *(atributo, valor)*, los patrones complementarios se pueden producir simplemente negando la condición. En un entorno multi-relacional, la producción de patrones complementarios de este tipo no es tan directa. Por ejemplo, si estamos considerando personas y la estructura de su hogar, y hemos establecido que el conjunto de personas que tienen al menos un hijo es interesante, podríamos querer dividir este conjunto en personas que tienen un hijo varón, y el complementario de ese conjunto. Claramente, este complementario no es igual al conjunto de personas que tienen una hija, ya que esto no excluye a las personas que tienen hijos e hijas. Es por ello que en el capítulo 3 se presentaron una serie de refinamientos que generan PQG complementarios. Nosotros usaremos esos refinamientos como

conjunto $REFS$ en la llamada del algoritmo en los ejemplos que se mostrarán a continuación, pero podría enriquecerse con cualquier refinamiento adicional que se considerase de valor.

4.3.1. Ejemplo de aplicación del algoritmo PQG-ID3

Vamos a presentar un caso concreto de aplicación del algoritmo PQG-ID3 sobre un pequeño grafo con propiedades a modo de demostración. Los refinamientos serán los vistos en el capítulo anterior; usaremos la condición de parada más restrictiva, es decir, que todos los pares del nodo actual pertenezcan a la misma clase; y se ha seleccionado la Ganancia de Información como medida de impureza.

Trabajaremos con el grafo social mostrado en la Figura 4.7, en el que se representan algunas conexiones de tipo marital entre usuarios, y otras relacionadas con la publicación de fotografías por estos usuarios. Podemos encontrar nodos de tipo `user` y `photo`, y relaciones de tipo `husband`, `wife`, `publish` y `likes`. No haremos uso de propiedades topológicas que hacen uso de medidas en el grafo, por lo que en el caso del refinamiento *añadir predicado a nodo* los predicados disponibles serán $\{type = photo, type = user\}$, y en el caso del refinamiento *añadir predicado a arista*, los predicados disponibles serán $\{type = publish, type = likes, type = husband\}$.

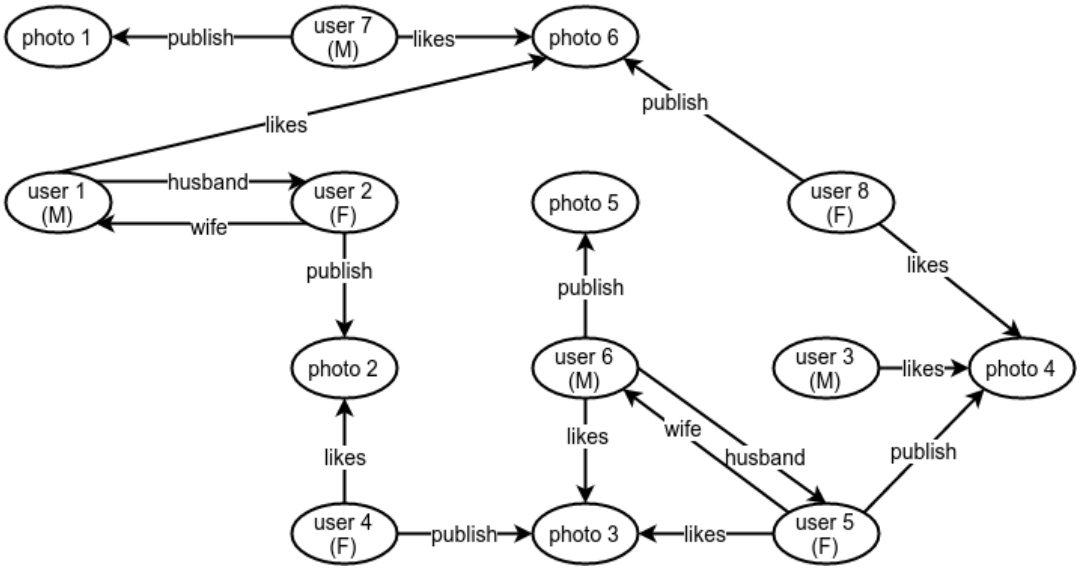


Figura 4.7: Grafo Social.

Adicionalmente, tenemos una propiedad en algunos nodos (los de tipo `user`) que indica su género, con posibles valores `F` (female) y `M` (male), los nodos de tipo `photo` tendrán asociado un valor de género `None`. Predecir este atributo constituirá el problema de clasificación sobre nodos. Aunque como hemos indicado el algoritmo PQG-ID3 está preparado para construir árboles de decisión capaces de clasificar cualquier estructura (subgrafo) en un grafo con propiedades, con el fin de mostrar un primer ejemplo simple, abordamos un problema que solo intenta clasificar nodos. Con el objetivo de que la exposición resulte más clara, y para evitar confusiones con la terminología, a los nodos pertenecientes al conjunto de entrenamiento los denominaremos *objetos*, mientras que dejaremos el término *nodos* para los nodos del árbol de decisión en construcción.

El algoritmo PQG-ID3 construye el árbol de decisión de la Figura 4.8 (los nodos/aristas positivas son marcados en negro y los negativos en rojo) que es capaz de clasificar correctamente todos los nodos del grafo social de la Figura 4.7 según su género: `male`, `female` o `None`. Todas las indicaciones que se hagan acerca de la construcción siguen la representación mostrada en la citada figura.

El conjunto de entrenamiento lo forman todos los nodos del grafo. En la ejecución del algoritmo el PQG inicial, Q_0, está compuesto por dos nodos positivos, uno de ellos con un predicado que exige pertenencia al subgrafo bajo evaluación ($v \in S$) y otro que exige la condición contraria ($v \notin S$). Como se ha comentado anteriormente, es habitual que el PQG inicial corresponda a la mayor subestructura común en los subgrafos a clasificar, en este caso, la mayor estructura común está compuesta por un único nodo sin restricciones. Además, y por motivos de eficiencia computacional, en cada paso del algoritmo se creará un nodo positivo aislado con un predicado que exige la no pertenencia al subgrafo bajo evalacuión ($v \notin S$) si es que no existe ningún nodo aislado de este tipo en el PQG actual. Si el subgrafo bajo evaluación no cubre todos los nodos del grafo en el que se encuentra inmerso (lo cual se cumple en todos los ejemplos presentados), añadir un nodo de este tipo no modifica la semántica asociada al PQG al que es añadido, como vimos en el capítulo anterior al analizar este tipo de refinamiento.

Como primer paso, el algoritmo analizará qué refinamiento de $REFS$ (y con qué parámetros) aporta una mayor ganancia de información, dando como resultado el refinamiento $+\{\xrightarrow{e^+}\}$ entre los únicos dos nodos existentes en Q_0. Por lo que el primer nodo (raíz) del árbol de decisión será el que realice el test sobre la existencia o no de una arista saliente en alguno de los nodos en el subgrafo a evaluar. Aunque, como vimos en el capítulo anterior, el refinamiento *añadir arista* genera cuatro PQG que refinan a Q_0, consideramos solo aquellos que intervienen en el proceso de clasificación de objetos, y no los que transmiten un conjunto vacío de objetos.

Debido a que los subgrafos que se quieren clasificar están compuestos por un único objeto, este refinamiento evalúa la existencia o no de una arista saliente en este objeto. Las ramas correspondientes a la existencia (respectivamente, no existencia) de dicha relación trasmitirán los objetos que posean (respectivamente, no posean) una relación saliente. De acuerdo a la representación dada en la Figura 4.8, por la rama derecha se trasmitirán todos los objetos que no posean una relación saliente (en este caso concreto, todos los objetos del grafo de datos de tipo `photo` y ninguno de tipo `user`), ninguno de estos objetos poseen valor asociado a la propiedad `genre` por lo que directamente esta rama queda asociada al valor de salida `None` y genera una hoja del árbol de decisión (ya que presenta pureza máxima respecto de la clasificación buscada). Por la rama izquierda se trasmitirán todos los objetos que posean una relación saliente (en este caso, todos los objetos del grafo de datos de tipo `user` y ninguno de tipo `photo`).

Como los valores de la propiedad `genre` no son homogéneos para estos objetos (este nodo del árbol de decisión actual presenta impureza), el algoritmo no acaba y es necesario aplicar un nuevo refinamiento a esta rama que produzca alguna ganancia de información adicional. Recordemos que hasta este nodo han llegado los objetos que reflejan usuarios masculinos y femeninos con una arista saliente. De nuevo, se debe evaluar qué refinamiento aporta una mayor ganancia de información.

El refinamiento $+\{\overset{e \wedge \{type=publish\}}{\longrightarrow}\}$, *añadir un nuevo predicado a la arista saliente*, es el que mayor ganancia de información aporta. De nuevo, aunque este refinamiento de añadir propiedad a arista entre nodos positivos genera cuatro PQG que refinan, representamos solo aquellos que intervienen en el proceso de clasificación de objetos, y no los que transmiten un conjunto vacío de objetos.

Los refinamientos aplicados a este nodo del árbol de decisión discriminarán qué objetos (de los que le llegan, que son los que tienen una arista saliente) verifican que esta arista es de tipo `publish` y cuáles no. De acuerdo a la representación dada en la Figura 4.8, por la rama derecha de dicho nodo se transmitirán los objetos que no tengan una arista saliente de tipo `publish`. En el caso que estamos clasificando, todos estos objetos son usuarios de género masculino, por lo que se alcanza la condición de parada de máxima pureza y el nodo pasa a ser una hoja del árbol de decisión asociada a la clase `male`. Por la rama izquierda del nodo se transmitirán los objetos del grafo de datos que tengan una arista saliente de tipo `publish`. En este caso, de nuevo los valores de la propiedad `genre` no son homogéneos (presentan impureza) por lo que el algoritmo debe continuar buscando un nuevo refinamiento en esta rama.

Procedemos, pues, a repetir el procedimiento a este nodo, al que han llegado los objetos que se corresponden con usuarios masculinos y femeninos con una

arista saliente de tipo `publish`. El refinamiento que mayor ganancia de información aporta es $+\{\xrightarrow{e^+}\}$, *añadir una arista* entre el nodo aislado ($v \notin S$) y el nodo destino de la relación tipo `publish`. De nuevo, tomamos en cuenta solo aquellos refinamientos por los que se transmiten objetos del grafo de datos original. La interpretación a este nivel del árbol de decisión es discriminar, entre los objetos que han publicado algo, aquellos cuya publicación recibe alguna arista entrante por otro objeto que no pertenece a la estructura bajo evaluación y los que no.

De las dos ramas que producen algún tipo de filtrado efectivo, por la derecha (siempre según la representación de la Figura 4.8) se transmitirán los objetos que no cumplan con dicha condición. Todos estos objetos son usuarios de género masculino (y representan usuarios que han publicado una foto que no le gusta a nadie), por lo que se alcanza la condición de parada (máxima pureza) y el nodo se convierte en una hoja del árbol de decisión asociada a la clase `male`. Por la rama izquierda se transmitirán los usuarios que hayan publicado una fotografía y les haya gustado a alguien (esta fotografía publicada tiene una arista entrante que no proviene del nodo bajo evaluación). Todos estos usuarios son mujeres por lo que se verifica también la condición de parada y la hoja producida queda asociada a la clase correspondiente.

De esta manera, hemos conseguido construir un árbol de decisión que es capaz de clasificar correctamente todos los objetos en el grafo de datos asignándolos correctamente a la clase a la que pertenecen según su género haciendo uso de la estructura multi-relacional en la que se encuentran inmersos.

Además, la interpretación de los diversos nodos del árbol de decisión muestra claramente cómo, por medio de los Property Query Graphs y el sistema de refinamientos construido, se pueden conseguir refinamientos que evalúan propiedades del contexto diferenciando entre los objetos que deben estar dentro de la estructura analizada y aquellos que deben estar fuera, ampliando considerablemente la capacidad expresiva del sistema de consulta y, en consecuencia, su capacidad discriminadora.

4.4. Algunos Ejemplos Representativos

A continuación presentamos algunos ejemplos de árboles de decisión multi-relacionales que hacen uso de PQG y que han sido obtenidos siguiendo el algoritmo PQG-ID3 presentado. Al igual que en la ejecución correspondiente al ejemplo paso a paso, en las ejecuciones correspondientes a dichos ejemplos se han ido añadiendo nodos positivos aislados no pertenecientes al subgrafo bajo evaluación en cada paso si no existían en el PQG y se ha partido de PQG ini-

ciales que contenían un nodo positivo con un predicado que lo fija al subgrafo bajo evaluación y otro nodo positivo con la restricción contraria. Los ejemplos han sido extraídos de bases de datos en grafo pequeñas pero con la suficiente complejidad como para mostrar la capacidad de descubrimiento de patrones que posee el algoritmo PQG-ID3. Además, en esta sección, no mostraremos los árboles completos resultantes (debido a que su tamaño es grande para ser mostrados aquí directamente) sino que en algunas ocasiones sólo mostraremos las hojas clasificadoras pertenecientes a los mismos y, en otras, algunas de las ramas más interesantes.

4.4.1. StarWars

El primer ejemplo de árbol de decisión multi-relacional obtenido a través del algoritmo PQG-ID3 lo conseguimos minando el grafo presentado en la Figura 3.11 con información sobre StarWars presentado en el capítulo anterior.

Los PQG presentados en la Figura 4.9 permiten discriminar cada personaje presente en el grafo según si es devoto del Imperio, de la Rebelión o de ninguno de los dos bandos. Se corresponden con las diversas hojas clasificadoras del árbol de decisión calculado automáticamente. Para ello, los nodos de tipo `institution` (junto con las aristas en las que participan) han sido eliminados de dicho grafo. Las posibles clases en las que clasifica el árbol obtenido son: `empire`, `rebellion` o `None` (en el caso en el que el personaje no sea devoto de ninguna de las dos instituciones).

Los PQG obtenidos en las hojas muestran que incluso trabajando con grafos relativamente pequeños el nivel de complejidad que pueden alcanzar las consultas proporcionan ejemplos muy interesantes de aprendizaje de patrones de forma automática usando la metodología presentada.

4.4.2. El Hobbit

El segundo ejemplo de árbol de decisión multi-relacional obtenido a través del algoritmo PQG-ID3 lo obtenemos minando otro grafo de juguete habitual en las pruebas realizadas para bases de datos en grafo, en este caso uno relacionado con la historia del Señor de los Anillos. Este grafo contiene 105 nodos distribuidos a través de 7 tipos de nodos (`Character`, `Event`, `Item`, `Clan`, `Aligment`, `Location` y `Chapter`) y 209 aristas distribuidas a través de 65 tipos. Un subgrafo de ejemplo extraído de dicha base de datos se muestra en la Figura 4.10.

Es un grafo interesante para hacer pruebas de aprendizaje de estructuras porque presenta una tipología en aristas muy elevada, con muy pocos represen-

tantes de algunos tipos de aristas, por lo que mecanismos ineficientes tenderán a crear árboles muy grandes como único método para poder realizar clasificaciones multi-relacionales.

Figura 4.10: Sección del grafo El Hobbit.

El árbol presentado en la Figura 4.11 permite discriminar cada ubicación (`Location`) presente en el grafo según su topografía, donde las posibilidades son `Hills`, `Forest`, `Valley`, `Mountain`, `Caves` y `Lake`. Para ello, dicha propiedad ha sido eliminada de los nodos para no poder ser utilizada en el aprendizaje. En la construcción del árbol se ha impuesto una profundidad máxima de 5 niveles y se han eliminado algunas ramas por motivos de presentación.

4.5. Algunas Notas sobre la Implementación

La implementación del algoritmo PQG-ID3 ha sido llevada a cabo en el lenguaje de programación Python. Como sistema de persistencia ha sido utilizada la base de datos en grafo Neo4j (por motivos similares a los expuestos en capítulos anteriores). La verificación de un PQG por parte de una estructura inmersa en Neo4j ha sido llevada a cabo haciendo uso del lenguaje de consultas Cypher, debido principalmente a que en dicho lenguaje las evaluaciones relacionadas con la existencia de caminos son muy expresivas y están altamente optimizadas.

Esta implementación ganaría en eficiencia si, en lugar de haber desarrollado el sistema de consulta sobre el lenguaje Cypher, se hubiera diseñado utilizando

la API Java de Neo4j o incluso implementando un sistema de persistencia *ad hoc* orientado a soportar este tipo de tareas. Pero, como hemos indicado anteriormente, nuestro objetivo en esta etapa es la de demostrar que el sistema formal de consultas es capaz de realizar esta tarea de forma sencilla e informativa para el usuario.

Se aprovecha la complementariedad de los PQG resultantes de un refinamiento para ahorrar tiempo en las consultas y, además, las hojas a las que no llega ningún elemento del conjunto de entrenamiento son podadas directamente.

Por la forma en que se construyen los refinamientos, y como se almacenan en el árbol de decisión, si un objeto verifica un PQG, también verificará todos los PQG antecesores de éste, por lo que la potencia clasificadora de un árbol de decisión obtenido a través del algoritmo PQG-ID3 está contenida en las hojas del árbol. Sin embargo, para ganar eficiencia, a la hora de clasificar un ejemplo nuevo a partir del árbol obtenido, se aconseja usar el árbol completo, ya que solo serán necesarias, a lo sumo, k evaluaciones (con k la profundidad del árbol), pero puede haber una cantidad exponencial de hojas clasificadoras. Además, como de un nivel al siguiente se ha producido una modificación atómica, aportada por el refinamiento correspondiente, la evaluación de un subgrafo en un nodo del árbol de decisión supone considerar únicamente algunas comprobaciones adicionales a la evaluación de su nodo padre.

4.6. Random Forest a partir de PQG-ID3

Como se ha comentado a lo largo de este trabajo, los árboles de decisión constituyen una herramienta idónea para ser combinada a través de algún modelo *ensemble* como Random Forest. Los motivos que hacen que los árboles de decisión sean adecuados para ser utilizados de manera combinada son el bajo coste computacional en el entrenamiento y la aletoriedad conseguida en el modelo a partir de pequeños cambios en el conjunto de datos de los que aprender.

Otra caractertística importante de los métodos de aprendizaje basados en árboles de decisión es que representan un modelo de caja blanca, ofreciendo una explicación interpretable por un humano de las decisiones tomadas a la hora de realizar regresión o clasficación sobre un dato. Esta última característica de los modelos de aprendizaje basados en árboles de decisión puede difuminarse cuando combinamos varios árboles para llevar a cabo una tarea de decisión.

El hecho de que perdamos capacidad en la interpretación de un resultado al combinar árboles multi-relacionales no impide que algoritmos como PQG-ID3 puedan ser combinados en este tipo de métodos agregados para llevar a cabo tareas de predicción de tipo caja negra. Sin embargo, existen posibilidades para

combinar diferentes árboles de este tipo y que sigan ofreciendo justificaciones interpretables por los humanos. Una posibilidad es combinar los PQG de hojas aociadas a la misma clase en los diferentes árboles, dando lugar a patrones combinados (posiblemente de manera probabilística) que son capaces de condensar los diferentes predicados que caracterizan a una misma clase en un predicado más potente.

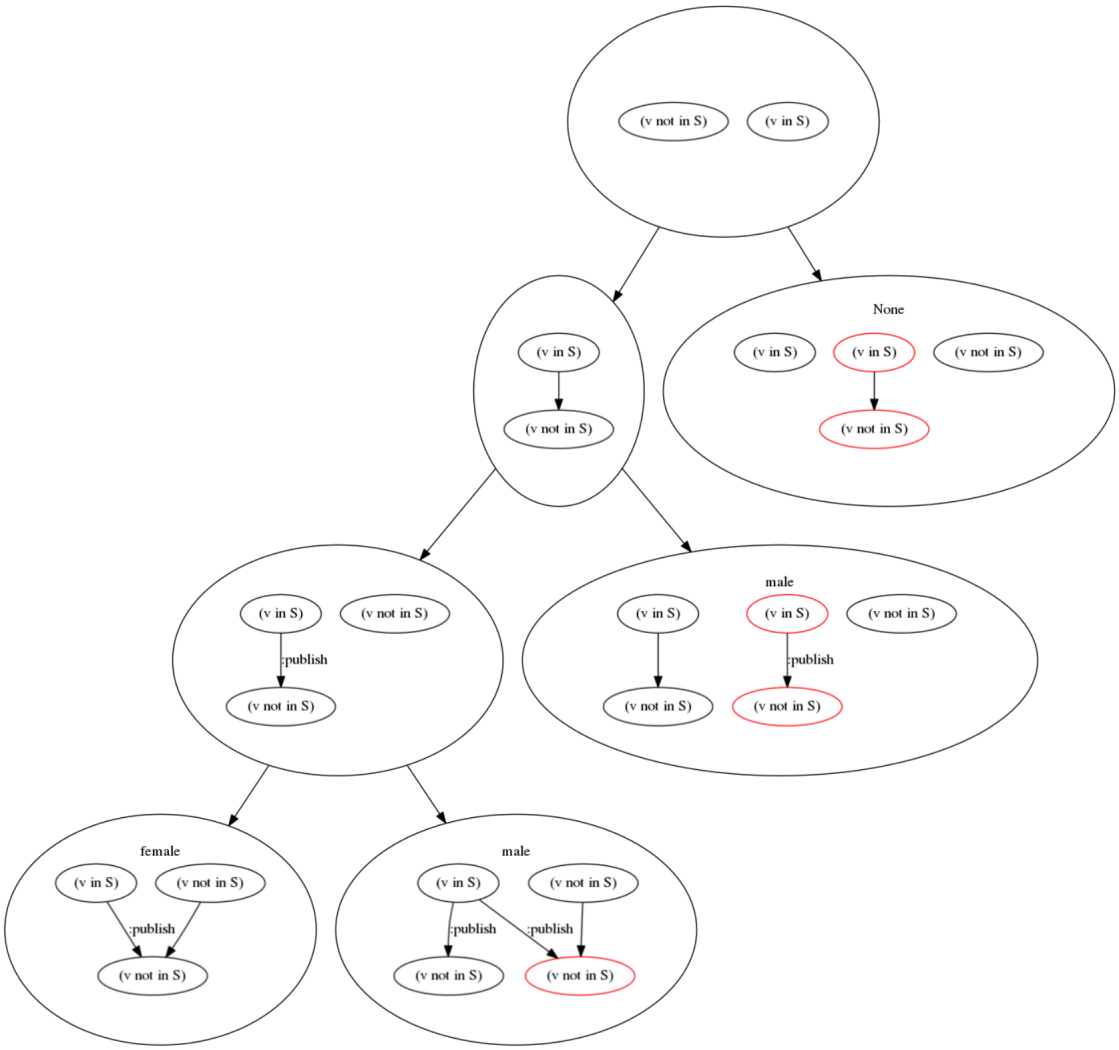


Figura 4.8: Árbol de decisión PQG (grafo social).

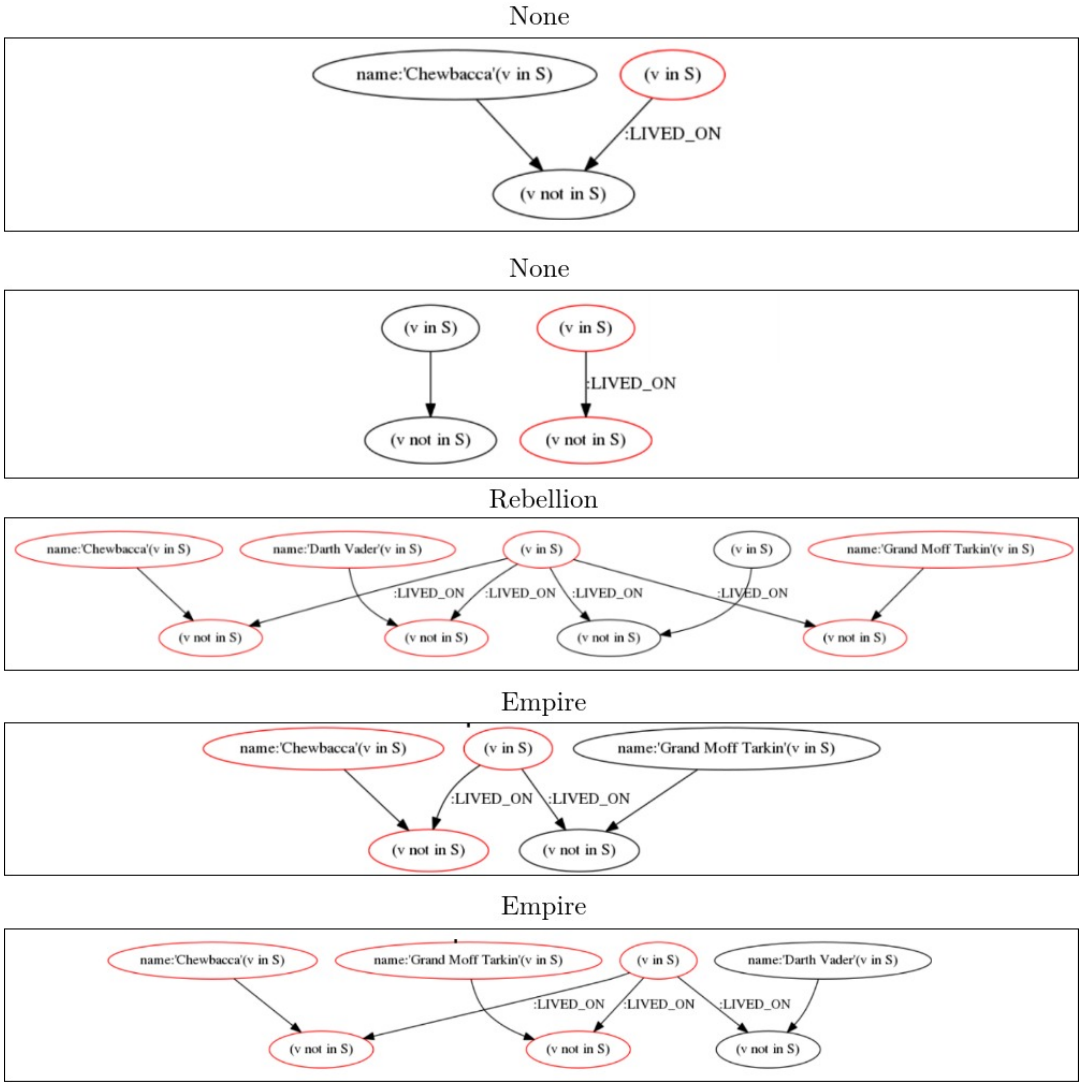


Figura 4.9: Hojas del árbol de decisión PQG (grafo Starwars)

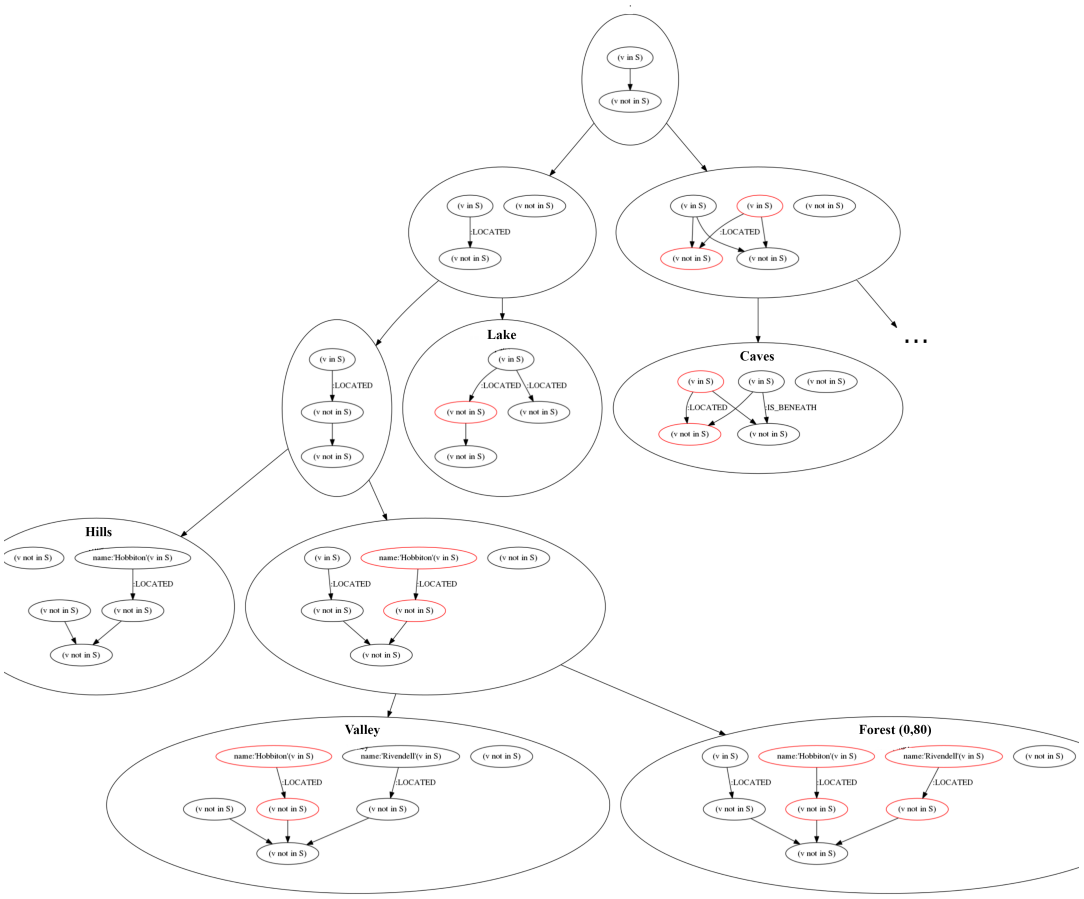


Figura 4.11: Árbol de decisión PQG (grafo El Hobbit).

Capítulo 5

Inmersión semántica de grafos con propiedades

En este último capítulo presentamos una nueva aproximación al tratamiento de Grafos con Propiedades, en este caso haciendo uso de técnicas derivadas del aprendizaje automático por medio de redes neuronales codificadoras. Concretamente, trataremos aquí el problema de la inmersión de Grafos con Propiedades en espacios vectoriales.

Durante este capítulo utilizaremos el término inmersión como una operación que permite considerar una estructura matemática, X, dentro de otra estructura Y, y que vendrá dada por una función inyectiva que preserve su estructura, $f : X \to Y$. En el caso que nos ocupa no nos interesa cualquier inmersión, ya que la mayoría de ellas no mantendrán las estructuras topológicas (dadas por las relaciones existentes en el grafo) y semánticas (dadas por los tipos asociados a tales relaciones) que confieren su interés al tipo de grafos que es nuestro foco de estudio, sino solo aquellas que sean capaces de reflejar, dentro de las características propias de un espacio vectorial (relaciones de distancia, linealidad y clusterización), las características interesantes del grafo.

Por ejemplo, sería interesante conseguir inmersiones que, al proyectar los nodos del grafo en puntos de un espacio vectorial, hagan que las relaciones existentes queden proyectadas automáticamente en vectores que mantengan los tipos. De esta forma, interpretamos que la carga semántica asociada a la relación ha sido capturada por la inmersión del grafo en el espacio vectorial. Otra opción es comprobar si la inmersión verifica propiedades de clusterización respecto a los tipos de nodos, sus propiedades, o algunas de las métricas que podemos definir sobre el grafo.

Posteriormente, haremos uso de estas buenas propiedades de inmersión para

ver si somos capaces de obtener herramientas de predicción / clasificación / descubrimiento en la red semántica original.

Para ello, y aunque dimos una breve introducción de las redes neuronales artificiales como modelos de aprendizaje en el capítulo de Fundamentos, especificaremos ahora su uso como *máquinas codificadoras*, y presentaremos nuestra propuesta de inmersión por medio de un codificador neuronal que extrae un conjunto de entrenamiento de las características de un grafo con propiedades.

Acabaremos el capítulo presentando algunos de los experimentos que hemos realizado haciendo uso de nuestra propuesta y viendo qué posibilidades abre para el análisis semántico de grandes cantidades de información.

5.1. Redes Neuronales Artificiales

Las *Redes Neuronales Artificiales* son un modelo matemático inspirado en el comportamiento biológico de las neuronas y en cómo éstas interactúan entre sí en el cerebro. Los primeros algoritmos inspirados en este tipo de sistemas fueron desarrollados a mediados del siglo XX por W. McCulloch y W. Pitts [146], y más de medio siglo de desarrollo práctico e importantes resultados teóricos han proporcionado unos fundamentos robustos que los han convertido en una herramienta fundamental en aplicaciones de Ingeniería, y uno de los modelos más usados dentro del Aprendizaje Automático. Hoy en día, y tras diversas modificaciones y mejoras desde las primeras propuestas, las redes neuronales artificiales son modelos adecuados para resolver problemas en los que la clasificación, regresión, o codificación son fundamentales para dar soluciones.

Uno de los primeros modelos matemáticos que se dieron de una neurona con fines computacionales fue el *Perceptrón* [205], que en una primera instancia funciona como un clasificador binario muy simple. El perceptrón recibe un conjunto de entradas numéricas y, tras un proceso de combinación lineal de sus entradas y discretización, devuelve un resultado binario. Podemos representar gráficamente su funcionamiento por medio de la Figura 5.1, donde el esqueleto mínimo del modelo intenta capturar las partes fundamentales de una neurona real:

- un *conjunto de sinapsis*, que le permiten recibir información (entradas) del exterior, que es traspasada a
- un *núcleo*, donde se procesa la información, y se manda a
- una *salida*, por la que la neurona devuelve una respuesta a los impulsos recibidos.

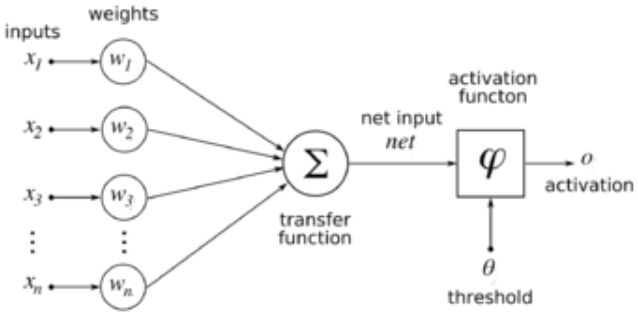


Figura 5.1: Arquitectura de un perceptrón.

Los parámetros libres del perceptrón original son los diversos pesos asociados a los canales de entrada, w_i, y un umbral que determina el comportamiento de respuesta según la intensidad alcanzada en su núcleo, θ. Obsérvese que el funcionamiento del núcleo se ha separado en dos procesos secuenciados: por una parte, la agregación de las entradas (representado por Σ), y por otra, la activación de la salida (que viene dada por la acción de una función φ). Aunque en el modelo original esta función de activación estaba restringida a algunas funciones básicas discretizadoras (como la función escalón), en los modelos posteriores se convierte en una opción más de diseño.

Más formalmente, si denotamos el vector de entradas por $\vec{x}$ y el vector de pesos por $\vec{w}$, el resultado del cálculo de un perceptrón a partir de sus entradas se calcula haciendo la suma ponderada, a través de los pesos $\vec{w}$, de dichas entradas, seguido de una función de activación que recibe dicha suma y ofrece como salida un 1, si la suma supera el umbral θ, o un 0, en caso contrario. De esta forma, el perceptrón calcula una salida:

$$o = \begin{cases} 1 & \text{, si } \vec{x}\vec{w} \geq \theta \\ 0 & \text{, si } \vec{x}\vec{w} < \theta \end{cases}$$

Con el fin de uniformizar la notación, suele ser habitual tratar el valor del umbral, θ, a través de una entrada adicional constante, $x_b = 1$, con un peso asociado $w_b = -\theta$ (al que llamamos *bias*) de tal manera que podemos representar la nueva entrada x_b dentro del vector $\vec{x}$, y el nuevo peso w_b dentro del vector de pesos $\vec{w}$ como un elemento más:

$$\vec{x} = \left[x_1, \ldots, x_n, x_b\right], \ \vec{w} = \left[w_1, \ldots, w_n, w_b\right]$$

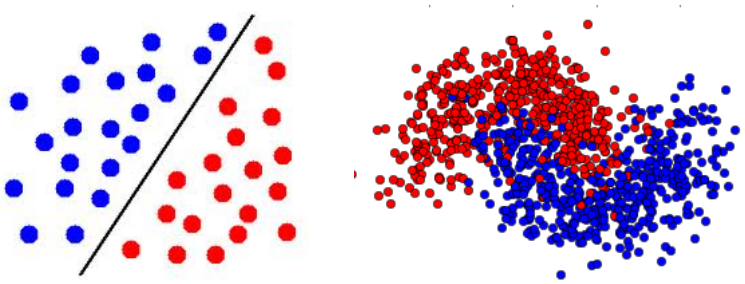
En consecuencia, la función que calcula el perceptrón se puede reescribir de forma sencilla haciendo uso de la función de Heaviside como función de activación:

$$h(\vec{x}) = H(\vec{x}\vec{w}) = \begin{cases} 1 & \text{, si } \vec{x}\vec{w} \geq 0 \\ 0 & \text{, en otro caso} \end{cases}$$

Debido a que el perceptrón así definido devuelve un resultado binario, suele ser común usarlo como clasificador binario, donde $h(\vec{x})$ es usado para clasificar $\vec{x}$ como una instancia positiva (si $h(\vec{x}) = 1$) o negativa (si $h(\vec{x}) = 0$).

Desde el punto de vista del aprendizaje automático, el problema de aprendizaje supervisado se convierte en encontrar los pesos adecuados para que el perceptrón devuelva la clasificación correcta para un conjunto de entrenamiento dado, y existen algoritmos sencillos que permiten (cuando es posible) encontrar los pesos adecuados de forma directa.

Es fácil ver las limitaciones que tiene este modelo, ya que solo si el conjunto de entrenamiento es linealmente separable (Figura 5.2(a)) se garantiza la convergencia del perceptrón con una función de activación Heaviside [171], pero en caso contrario (Figura 5.2(b)), es claro que un perceptrón de este tipo no podría clasificar adecuadamente los elementos del conjunto.



(a) Conjunto con dos clases linealmente separable.

(b) Conjunto con 2 clases no linealmente separable.

Una posible solución para conseguir modelos más flexibles se basa en considerar diferentes funciones de activación. En estos casos, o bien podemos clasificar correctamente conjuntos no linealmente separables, o bien podemos aproximar funciones no lineales en el caso de que estemos buscando un procedimiento de regresión. Pero incluso considerando estas modificaciones sigue siendo un modelo bastante limitado. La extensión natural para obtener modelos más potentes pasa por combinar varias neuronas para que las salidas de algunas de ellas se conviertan en las entradas de otras, de forma que podamos obtener mayor complejidad que la que un solo perceptrón proporciona.

Aunque la primera intención es combinarlas con absoluta libertad, tal y como aparecen en los seres vivos, las estructuras topológicas que siguen las neuronas de un sistema nervioso real son redes excesivamente complejas y no disponemos de las herramientas matemáticas adecuadas para asegurar su correcto funcionamiento, por lo que es habitual trabajar con estructuras topológicas más limitadas pero que proporcionan un entorno controlado en el que podemos dar resultados teóricos robustos acerca de su capacidad expresiva. En este

contexto, las redes más habituales suelen ser las *Redes Feedforward*, o *Redes Unidireccionales*, formadas por un conjunto de perceptrones interconectados de tal forma que las conexiones entre ellos no forman ciclos (no se produce el efecto de retroalimentación) y donde la función de activación puede ser cualquiera (no únicamente la función Heaviside). La formalización de estas redes (Figura 5.2) es la que se mostró en el capítulo 2.

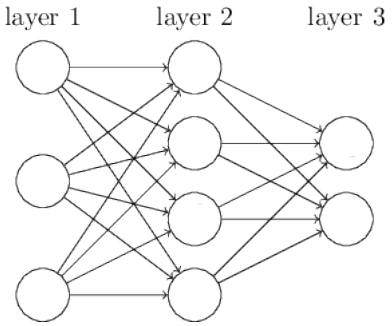


Figura 5.2: Red neuronal con una capa oculta.

Aunque estas redes parezcan muy restrictivas en cuanto a su complejidad estructural, el **Teorema de Aproximación Universal** [61] demuestra que una red neuronal feedforward con una única capa oculta conteniendo un número finito de neuronas, y bajo condiciones concretas de la función de activación, puede aproximar funciones continuas en subconjuntos compactos de $\mathbb{R}^n$. Formalmente, y para el caso de funciones reales, quedaría:

Teorema 15 (Teorema de Aproximación Universal)**.** *Sea σ una función de activación continua, monótona, no constante, y acotada (esto es, con $\lim_{x\to+\infty}\sigma(x) = c$, $\lim_{x\to-\infty}\sigma(x) = d$ y $c \neq d$), y $K \subseteq \mathbb{R}$ un compacto. Entonces, $\forall f \in C(K)$, $\forall\epsilon > 0$, $\exists n \in \mathbb{N}$, y $\vec{w}, \vec{a}, \vec{b} \in \mathbb{R}^n$ tal que:*

$$\sup_{x\in K} |f(x) - \sum_{i=1}^{n} w_i\sigma(a_i x + b_i)| < \epsilon$$

Sin embargo, este teorema de densidad no proporciona un algoritmo, $\mathcal{A}$, que permita obtener la red (los parámetros del modelo) que asegura cierta aproximación a la función objetivo, únicamente asegura su existencia.

Como suele ser habitual cuando se buscan resultados de aproximación, tenemos la opción de encontrar un buen aproximador haciendo uso de los algoritmos iterados de optimización habituales que intentan minimizar el error cometido entre la función objetivo y el cálculo actual de nuestro candidato. De entre estos métodos, uno de los más comunes es el del *Descenso del Gradiente*, donde por medio de las derivadas (o derivadas parciales) somos capaces de modificar los

parámetros del modelo para conseguir sucesivas reducciones del error cometido. Por ello, han de tenerse en cuenta varias consideraciones:

- En primer lugar, para poder aplicar este método es necesario que el cálculo de la función de salida se pueda expresar como una función derivable de los parámetros del modelo (los pesos), lo que nos lleva a trabajar con funciones de activación que, además de las condiciones que impone el teorema anterior, sean derivables.

- En segundo lugar, debe recordarse que, en general, estos métodos no aseguran la obtención de un óptimo global, por lo que puede ocurrir (y con frecuencia ocurre) que nos quedemos atrapados en un óptimo local que no aproxima suficientemente bien nuestra función objetivo.

La implementación más habitual de este algoritmo para el entrenamiento de redes feedforward se conoce como *algoritmo de retropropagación* (o de *propagación hacia atrás*). A pesar de su buen funcionamiento en muchísimos casos, es conveniente hacer algunos comentarios que pueden arrojar luz acerca de la evolución que ha sufrido el campo de redes neuronales hasta la actualidad y que justifican algunas de las decisiones que hemos tomado en nuestra aproximación:

- Las modificaciones que este algoritmo realiza sobre los pesos de conexiones que provienen de neuronas con una baja activación son menores que las realizadas sobre aquellas que provienen de una neurona con un alto valor de activación (se dice que el aprendizaje realizado sobre estas conexiones es menor).

- La elección de la función de activación es esencial a la hora de conseguir que una red aprenda una función determinada de manera eficiente (aunque el Teorema de Aproximación Universal confirma la existencia de parámetros adecuados para una buena aproximación, no asegura que podamos acercarnos a los valores óptimos haciendo uso de gradientes). Si utilizamos una función lineal como función de activación no podremos conseguir que la red aprenda una función que no sea lineal, ya que la combinación lineal de funciones lineales da como resultado una función lineal. De hecho, aunque las funciones lineales son continuas y derivables, no están en las condiciones del Teorema de Aproximación Universal, ya que no están acotadas, por lo que no aseguran una correcta aproximación. Por otra parte, la función Heaviside sí está acotada y verifica las condiciones del Teorema de Aproximación Universal, pero no es continua, y por tanto no es derivable, por lo que a pesar de existir aproximaciones buenas usándola, no podemos usarlas en el método del gradiente (de hecho,

allí donde es derivable, la función Heaviside tiene derivada nula, por lo que no podríamos ajustar los parámetros haciendo uso del algoritmo de retropropagación).

- Una función muy adecuada para ser utilizada como función de activación es la función *sigmoide* (o *logística*) (ver Figura 5.3):

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

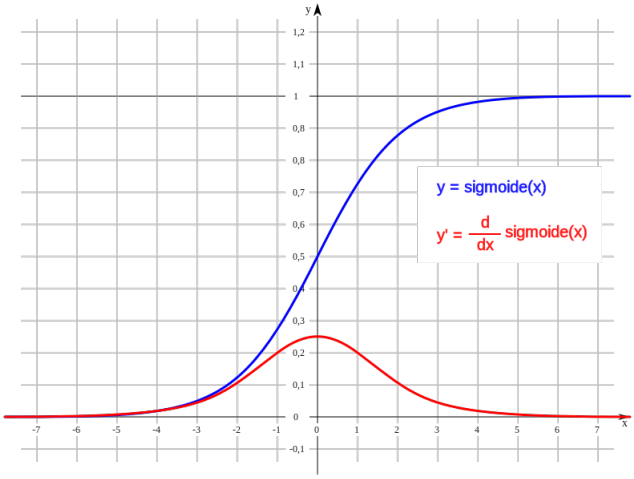


Figura 5.3: Función sigmoide y derivada.

Por una parte, está acotada, con $(0, 1)$ como rango de salida, lo que hace que esté en las condiciones del Teorema de Aproximación Universal. Por otra parte, es derivable, por lo que está en las condiciones para usar un algoritmo de retropropagación basado en el descenso del gradiente, y la derivada de la función sigmoide es muy sencilla, y también está acotada:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

lo que implica que, una vez que se ha calculado $\sigma(x)$, el cálculo de $\sigma'(x)$ sea inmediato, permitiendo que los cálculos en la fase de retropropagación sean más rápidos (basta almacenar en cada neurona el valor de su activación en el paso de propagación, para ser reutilizado en el paso de retropropagación).

- Pero una función como la función sigmoide también presenta algunos inconvenientes a la hora de ser utilizada como función de activación. Si la activación de una neurona en la capa de salida está cercana a 0 o a 1 la

derivada de ésta es prácticamente nula y, por tanto, el aprendizaje de los pesos de las conexiones que llegan a ella se producirá de una manera muy lenta. Esta situación puede ocurrir cuando la entrada total de la neurona es muy grande en valor absoluto, en este caso se dice que la neurona está *saturada*. Además, la derivada de la función sigmoide es relativamente pequeña en todos sus puntos (siempre por debajo de 0,3), por lo que el aprendizaje en las capas más alejadas de la capa de salida es casi nulo (ya que en ellas interviene el producto acumulado de los valores de aprendizaje de las capas posteriores). Este es el motivo por el que no suele ser posible entrenar por retropropagación redes neuronales con un número de capas elevado y ha sido necesario buscar métodos alternativos de aproximación o estructuras distintas a las feedforward para estos fines.

5.1.1. Codificadores

El uso más habitual de las redes neuronales feedforward ha sido como máquinas de cálculo, pero en esta sección presentamos un uso que será (y ha sido) de fundamental importancia para los nuevos resultados que se han obtenido con ellas.

Obsérvese que cuando una red feedforward tiene capas ocultas toda la comunicación que se produce entre la capa de entrada y la capa de salida pasa por cada una de las capas ocultas. De esta forma, si estamos intentando aproximar una función por medio de una red feedforward que tiene una capa oculta, tras el ajuste de los parámetros de la red (se haga por el procedimiento que se haga), podemos pensar que la capa oculta mantiene la información necesaria de los datos de entrada que son necesarios para el cálculo de la función. Por ello, siempre desde el punto de vista de la función que calcula la red, podemos decir que la capa oculta codifica los datos de entrada; y los pesos (y bias) que se han usado definen la función de codificación entre ambos espacios [108]. De igual forma, podemos entender que la parte de la red original que va desde la capa oculta que consideremos hasta la capa de salida define una decodificación hacia el espacio de llegada (ver Figura 5.4.

De forma general, el objetivo de los codificadores neuronales es aprender una codificación a partir de un conjunto de datos. Si prescindimos de las capas posteriores (incluida la capa de salida original) y de los parámetros que tienen asociados, obtenemos una nueva red neuronal que produce como salida una representación del espacio de entrada en un espacio de dimensión concreta (el número de neuronas en la capa oculta que se ha considerado). Debemos recordar que esta representación se consigue como aplicación parcial de una función que se ha obtenido a partir de una red feedforward completa que aproxima una

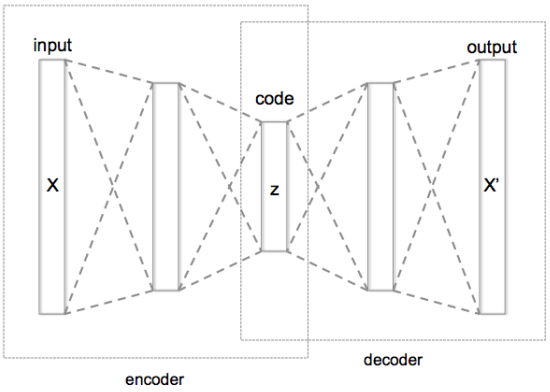


Figura 5.4: Codificador neuronal.

función prefijada y, consecuentemente, la codificación obtenida es relativa a esta función (y, por supuesto, al proceso de aproximación).

Los *autocodificadores* son un caso concreto de codificador neuronal en el que se ha intentado aprender la función identidad y, en consecuencia, las capas de entrada y salida poseen el mismo número de neuronas. Es decir, el conjunto de muestras de entrenamiento sería $S = \{(\vec{x}^1, \vec{x}^1), \ldots, (\vec{x}^N, \vec{x}^N)\}$. Al igual que ocurre cuando trabajamos con codificadores, cuando la red alcanza un estado aceptable (es capaz de mostrar una salida suficientemente parecida a la entrada para cada uno de los ejemplos), las activaciones en las unidades de las capas ocultas capturan información del dato original $\vec{x}$ presentado en la capa de entrada [27].

Si el número de unidades en la capa oculta usada para la codificación difiere del número de unidades en la capa de entrada (y salida) estaremos además haciendo un cambio dimensional al realizar la codificación. De hecho, es uno de los métodos que pueden usarse para realizar cambios de dimensionalidad manteniendo las características estructurales presentes en los conjuntos de entrenamiento (por ejemplo, las relaciones de proximidad o similitud).

Si los tamaños de las capas ocultas son los mismos que las capas de entrada y salida se deben imponer condiciones adicionales para asegurar que no estamos aprendiendo la función identidad de forma trivial.

Como hemos indicado, en este capítulo usaremos codificadores neuronales como medio de inmersión de nuestras estructuras de Grafo con Propiedades en un espacio $\mathbb{R}^n$ adecuado. Para ello, haremos uso de redes neuronales entrenadas con funciones adecuadas con el fin de comprobar hasta qué punto las estructuras semánticas del grafo se conservan en las propiedades vectoriales de la inmersión.

Pero antes presentaremos un modelo que ha servido de inspiración para nuestro trabajo y que realiza un proceso de inmersión de textos en espacios vectoriales manteniendo muchas propiedades de las estructuras gramaticales presentes en los textos originales.

5.1.2. Word2vec

La aplicación de los codificadores neuronales a textos ha proporcionado resultados muy interesantes. En 2013, T. Mikolov et al.[151] presentaron dos nuevas arquitecturas, denominadas bajo el nombre genérico de *word2vec*, para aprender representaciones vectoriales de palabras que tratan de minimizar la complejidad computacional a la vez que mantener muchas de las propiedades gramaticales presentes en los textos de donde se extraen: *Continuous bag-of-words* (CBOW) y *Skip-gram*.

El proceso para ambas arquitecturas pasa por una primera fase de preprocesamiento en la que se extrae la información de los textos de los que queremos aprender las estructuras. Para ello, se fija un vocabulario de trabajo extraído a partir de los textos que sirven de corpus para el aprendizaje. En muchas ocasiones, este vocabulario es procesado de forma que se trabaja únicamente con los lemas de las palabras (la raíz informativa de las palabras, eliminando los prefijos y sufijos que determinan su género, número, forma verbal, etc.), y es habitual trabajar únicamente con un subconjunto de palabras significativas, eliminando lo que se llaman *stopwords* (palabras vacías de significado, tales como preposiciones, pronombres, artículos, etc). El conjunto de palabras a considerar dependerá del problema de representación final que queramos obtener.

En esta tarea se define el *contexto de una palabra* en un texto como el conjunto de palabras que aparecen en posiciones adyacentes a ella. El tamaño del contexto, también llamado *ventana*, es variable y se decide en función de la codificación a realizar, por lo que se convierte en un parámetro libre de los modelos que puede ser ajustado para mejorar los resultados.

Las dos arquitecturas presentadas en [151] consisten en redes neuronales artificiales feedforward con 3 capas: una capa de entrada, una capa oculta (capa de proyección) y una capa de salida; pero se diferencian en la función objetivo que intentan aproximar. Por una parte, los codificadores neuronales con arquitectura CBOW toman el contexto de una palabra determinada como entrada y tratan de predecir dicha palabra en su salida. Por el contrario, los codificadores con arquitectura Skip-gram reciben la palabra como entrada y tratan de predecir el contexto asociado a ella en su salida. Las capas de entrada y de salida (de igual tamaño) tendrán tantas neuronas como palabras distintas tenga el vocabulario, de forma que si ordenamos el vocabulario, $V = \{w^1, \ldots, w^k\}$,

entonces la neurona i-ésima de esas capas codificará a w^i (es decir, se da una representación 1-aria del vocabulario) (Figura 5.5).

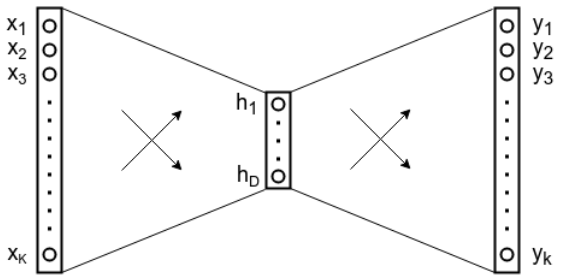


Figura 5.5: Arquitecturas CBOW y Skip-gram.

La capa de entrada de la red representa un vector de tipo *one-hot* que posee una neurona por cada palabra en el vocabulario de tal forma que, para el caso Skip-gram, la palabra w a ingresar activará (pondrá a 1) la única entrada asociada a ella en la capa de entrada y el resto de entradas permanecerán a 0, la red deberá aprender a mostrar por la salida el contexto C asociado a dicha palabra codificado también a través del vector *one-hot* en la salida de la red, de tal forma que se activen sólo las salidas asociadas a las palabras en dicho contexto C y el resto deberá permanecer a 0. En el caso de la arquitectura CBOW se ingresa el vector promedio de los vectores contexto asociados a una palabra determinada y la red deberá aprender a activar sólo la palabra asociada a dicho contexto en la capa de salida. Ver Figura 5.6. [204].

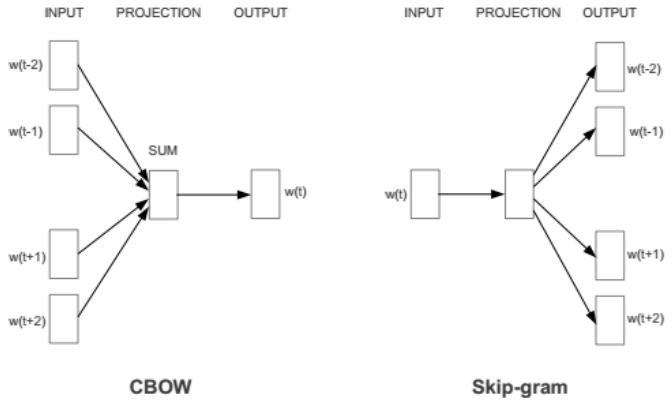


Figura 5.6: Arquitecturas CBOW y Skip-gram (detalle).

Formalmente, la tarea de entrenamiento supervisado se realizará con el conjunto de muestras de la forma (w, C) en el caso de la arquitectura Skip-gram, y

a través de un conjunto de muestras de la forma (C, w) en el caso de la arquitectura CBOW, donde w representa una de las palabras de V, y C representa uno de los posibles contextos asociados a dicha palabra. Debe tenerse en cuenta que la misma palabra puede (y normalmente así será) aparecer asociada a varios contextos, y viceversa.

Por ejemplo, dado el siguiente texto:

Creo en la vida eterna en este mundo, hay momentos en que el tiempo se detiene de repente para dar lugar a la eternidad.

En el caso de que consideremos las preposiciones y los artículos como *stopwords*, el vocabulario de dicho texto sería *[creer, vida, eterno, mundo, momento, tiempo, detener, repente, dar, lugar, eternidad]* y, si consideramos un tamaño de contexto 4, ejemplos de posibles pares (w, C) serían:

$$(eterno, [creer, vida, mundo, momento])$$

$$(mundo, [vida, eterno, momento, tiempo)$$

$$(momento, [eterno, mundo, tiempo, detener])$$

El objetivo principal del trabajo de Mikolov et al. es reducir la complejidad en el modelo neuronal para permitir al sistema aprender a partir de un gran volumen de datos textuales. Hasta la llegada de word2vec, ninguna arquitectura de este tipo había sido capaz de entrenarse con más de unos pocos millones de palabras.

Por medio de la relación que se establece entre las palabras del vocabulario y sus contextos, el modelo captura diferentes tipos de similaridad [154], tanto funcionales como estructurales, y proporciona una inmersión en el espacio vectorial que refleja estas similaridades. Por ejemplo, puede capturar similaridades entre palabras que juegan el mismo papel en las oraciones en las que intervienen [152] o incluso encontrar relaciones semánticas más complejas entre conceptos. Además, estas relaciones se proyectan en relaciones lineales en el espacio vectorial. Por ejemplo, si realizamos operaciones vectoriales entre las palabras (donde $+$ representa la suma habitual de dos vectores y $-$ su diferencia) se obtienen resultados como:

$$w2v(\text{"}Hombre\text{"}) - w2v(\text{"}Mujer\text{"}) + w2v(\text{"}Rey\text{"}) \approx w2v(\text{"}Reina\text{"})$$

donde $w2v$ es la codificación obtenida tras el entrenamiento, y que podríamos interpretar como

Hombre es a Mujer como Rey es a ... Reina

que demuestra la potencia de este tipo de codificaciones, siendo capaz de encontrar relaciones semánticas *subyacentes* tan sólo a partir de grandes volúmenes de textos y el adecuado entrenamiento.

5.2. Trabajos relacionados

En los últimos años se han desarrollado diferentes métodos que tratan de aprender representaciones vectoriales de entidades y relaciones en bases de conocimiento [90, 37, 223]. Todas ellas representan las entidades como vectores en un espacio vectorial determinado y las relaciones como combinación de las representaciones de las entidades que participan en ella.

En [37] se propone una inmersión de datos multi-relacionales en un espacio vectorial intentando verificar algunas propiedades adicionales. Concretamente, se busca una proyección de nodos y tipos de aristas, π, en el espacio vectorial buscando:

1. Minimizar la distancia $d(\pi(s) + \pi(l), \pi(t))$ de cada relación $(s \xrightarrow{l} t)$ existente (observada) en el conjunto de datos, donde s representa el elemento origen de la relación, l representa el tipo de relación, y t representa el elemento destino de la relación.

2. Maximizar las distancias $d(\pi(s')+\pi(l), \pi(t))$, $d(\pi(s)+\pi(l'), \pi(t))$ y $d(\pi(s)+\pi(l), \pi(t'))$, donde s' y t' representan nodos del grafo, y l' representa un tipo de relación del grafo, para los que las relaciones $s' \xrightarrow{l} t$, $s \xrightarrow{l} t'$ y $s \xrightarrow{l'} t$ no existen (relaciones no observadas) en el grafo.

Para mejorar la eficiencia del algoritmo, los autores hacen un muestreo aleatorio del grafo original, tanto para las relaciones existentes como para las no existentes. El proceso de optimización se realiza posteriormente siguiendo el método del descenso por el gradiente en modo *mini-batch* (es decir, por bloques de muestras de entrenamiento de tamaño prefijado, no todas en cada iteración). En [90], y con el fin de conseguir mejores resultados en la proyección, los autores siguen un procedimiento similar pero haciendo uso de una red neuronal siamesa[1] en vez de una red neuronal estándar. En [249] se agrupan algunas de estas técnicas sobre un mismo marco teórico general que permite la comparación tanto en complejidad de los modelos como en los resultados.

Pese a la relación existente entre estos trabajos y nuestra aproximación, el segundo requerimiento que imponen de maximizar las relaciones no observadas (y que es imprescindible para los resultados que obtienen) va en contra de uno de los objetivos que perseguimos, ya que no suponemos que el grafo original con propiedades tenga información completa y, en consecuencia, las relaciones no observadas pueden deberse a una carencia informativa y no a una inexistencia

[1] Una red neuronal siamesa es un tipo de red neuronal comparativa compuesta por dos redes que comparten pesos y arquitectura (cada una recibe un dato a ser comparado) y cuyas salidas son comparadas mediante una función de distancia.

real. Aún más, precisamente la predicción de este tipo de relaciones no observadas es una de las razones por las que buscamos una inmersión en un espacio que nos ofrezca una capacidad de análisis adicional.

Además, la mayoría de los trabajos que realizan aprendizaje de representaciones vectoriales de entidades y relaciones en bases de conocimiento tienen como objetivo evaluar la posibilidad de existencia de determinadas relaciones (*Link Prediction*), condicionando con un aprendizaje supervisado la representación de las entidades. En nuestro caso, la codificación que trataremos de aprender estará sólo condicionada por conseguir representaciones vectoriales que capturen la similitud de los contextos en los que se encuentran inmersas las entidades que representan, abriendo así la posibilidad de usar estas representaciones a un abanico más amplio de tareas.

DeepWalk [178] es una metodología reciente que utiliza codificadores neuronales para representar los nodos de un grafo uni-relacional haciendo uso de una idea muy similar a la que Mikolov et al. presentaron para la inmersión de grandes conjuntos de textos en espacios vectoriales. En concreto, en el citado trabajo el grafo uni-relacional es *linealizado* a partir de la generación de caminos aleatorios truncados, interpretando los caminos obtenidos como frases y considerando a partir de ellos un conjunto de entrenamiento con la forma

$$S = \{(n_1, C_1), \ldots, (n_N, C_N)\}$$

donde n_i representa un nodo concreto del grafo y C_i un contexto generado a partir de dichos caminos aleatorios truncados, es decir, el conjunto de nodos que aparecen en el mismo camino según el orden de recorrido. Posteriormente, y de forma completamente equivalente a word2vec, S es utilizado para entrenar un codificador neuronal con arquitectura Skip-gram y obtener así una inmersión de los nodos del grafo uni-relacional en un espacio vectorial. El método propuesto no permite trabajar con grafos multi-relacionales *grandes* de manera eficiente.

En [226] se presenta la metodología LINE para aprender una representación d-dimensional de los elementos inmersos en un grafo uni-relacional a través de dos fases: primero se aprenden $d/2$ dimensiones generando caminos aleatorios en modo Breath-First Search, posteriormente, se aprenden las $d/2$ dimensiones restantes haciendo un muestreo de los nodos que están estrictamente a distancia 2 del nodo origen.

Node2vec [95] agrupa y extiende las ideas presentadas en DeepWalk y LINE ampliando las posibilidades a la hora de construir los caminos aleatorios en el grafo. Concretamente, desarrollan un algoritmo flexible que, a través de dos hiperparámetros, permite modificar la generación de los caminos aleatorios que exploran el entorno de los nodos y dan lugar a su contexto. A partir de dos estrategias estándar de búsqueda, Breath-First Sampling (BFS) y Depth-First

Sampling (DFS), los dos parámetros permiten controlar si el camino aleatorio tiende a una estrategia BFS o DFS. En particular, afirman que un sampleo guiado por una estrategia BFS da lugar a inmersiones que reflejan la *equivalencia estructural* entre las entidades y que un sampleo guiado por una estrategia DFS da lugar a una inmersión en la que se refleja la *homofilia*. Por medio de epxerimentos se evalúa su capacidad para llevar a cabo *Multi-Label Classification* y *Link Prediction*. En este sentido, DeepWalk sería un caso concreto en el que el valor de ambos parámetros es el mismo.

En ninguno de los trabajos anteriores ([178, 226, 95]) se trabaja con grafos multi-relacionales, y se limitan las características a detectar la *homofilia* y la *equivalencia estructural* en las representaciones vectoriales.

En los últimos años han sido publicados algunos trabajos que hacen uso de Redes Neuronales Convolucionales (CNN) para crear representaciones vectoriales de los nodos de un grafo uni-relacional. En [125], el objetivo es aprender una función que codifique las características de los nodos de un grafo a partir de una descripción de las mismas (almacenadas en una matriz de descripciones) y de la matriz de adyacencias del grafo. También se pueden obtener salidas que representen el grafo completo aplicando alguna operación de tipo *pooling* [69]. En [65] sin embargo, se trabaja con el operador de convolución en el campo de Fourier, y generalizan las redes convolucionales para pasar de su definición original en espacios euclídeos regulares de baja dimensión (donde se trabaja naturalmente con imágenes, vídeos o audios) a poder trabajar con dominios irregulares de alta dimensión (grafos multi-relacionales obtenidos de redes sociales o de fenómenos biológicos). En [214] se presenta una extensión del modelo Graph Convolutional Network [125] denominado Relational Graph Convolutional Network, que permite realizar aprendizaje a través de las operaciones de convolución y *pooling* típicas de las redes convolucionales sobre grafos multi-relacionales.

En [212] se define el modelo Graph Neural Network Model, que convierte el grafo de datos en una red neuronal recurrente, y cada nodo del mismo en una red feedforward multi-capa. La combinación de estas estructuras permite llevar a cabo un aprendizaje en el que muchos de los pesos de la red son compartidos, reduciendo así el coste en el aprendizaje.

En [49] presentan HNE (Heterogeneous Network Embedding), un framework para hacer inmersiones de redes que conectan datos de diferentes tipos en espacios de baja dimensión utilizando una red neuronal profunda. Como el aprendizaje que se lleva a cabo es no supervisado, la nueva representación es adecuada para aplicar cualquier algoritmo de aprendizaje automático ya que no se ha condicionado el aprendizaje a una tarea determinada. Haciendo uso de estas representaciones, en [115] se enfrentan a la tarea de asignación automática de etiquetas a nodos de diferentes tipos en una red heterogénea que no tiene

tipos en las aristas. El algoritmo que presentan está diseñado para aprender las dependencias existentes entre los conjuntos de etiquetas asociadas a los diferentes nodos y para inferir las etiquetas asociadas a un nodo explotando las propiedades del grafo global y las caraceristicas de los nodos vecinos. Para ello, imponen dos objetivos: (1) intentar agrupar nodos del mismo tipo que estén conectados (con menos intensidad cuanto más largo sea el camino que los conecta), y (2) intentar agrupar nodos de diferentes tipos si comparten contextos. Para trabajar con grafos con propiedades, representan cada propiedad como un nodo nuevo.

5.3. Inmersiones de Grafos con Propiedades

La metodología que presentamos a continuación hace uso de un codificador neuronal, similar al usado en la arquitectura CBOW, para codificar los elementos de un grafo con propiedades en un espacio vectorial adecuado.

Aunque un grafo con propiedades tiene muchos elementos constitutivos, en una primera aproximación, y con el fin de evaluar hasta qué punto se mantiene la estructura semántica dada por las aristas, haremos una proyección usando únicamente el conjunto de nodos sobre el espacio vectorial. De esta forma, siguiendo con la analogía que nos ofrece el algoritmo word2vec, nuestro vocabulario será el conjunto de nodos del grafo (y también sus propiedades asociadas).

Un contexto, C, asociado a un nodo $n \in V$ se obtiene seleccionando, aleatoriamente y con repetición, un número determinado de nodos vecinos a n y propiedades suyas, independientemente del tipo de relación que los conecta y del tipo de propiedad. El número de nodos/propiedades seleccionados determina el *tamaño de la ventana de selección*.

Siguiendo una metodología similar a las vistas en el apartado anterior, generaremos un conjunto de entrenamiento formado por pares (n, C), donde $n \in V$ y C es uno de sus contextos asociados. El conjunto de muestras es utilizado para entrenar el codificador neuronal y, a continuación, las activaciones de la capa oculta de la red neuronal se utilizan como representación vectorial de cada uno de los nodos.

Una vez entrenado el codificador, usaremos estas representaciones vectoriales para intentar resolver algunas tareas de clasificación y descubrimiento en el grafo original. Los resultados de estas tareas nos darán una medida de fiabilidad sobre las inmersiones conseguidas (Figura 5.7). El Algoritmo 8 muestra la forma en la que obtenemos codificaciones vectoriales de cada vértice en un gafo con propiedades haciendo uso de una arquitectura tipo $CBOW$.

En el procedimiento de inmersión los *parámetros libres* del modelo, que habrá

Algorithm 8 Graph2Vec(G,N,ws,D)

```
training_set = {}
for each 0 < i ≤ N do
    n = randomly selected element from V_G
    C = {}
    for each 0 < i ≤ ws do
        e = randomly selected element from 𝒩(n) ∪ μ_G(n)
        C = C ∪ {e}
    end for
    training_set = training_set ∪ {(v, C)}
end for
Train a CBOW-like architecture with D neurons in hidden layer using training_set
return The resulting encoding for each element in V
```

que ajustar en los diversos experimentos para analizar su eficacia y viabilidad, son:

- D, tamaño de la capa oculta, determina la dimensión del espacio vectorial de inmersión en el que proyectaremos los elementos del grafo.
- N, tamaño del conjunto de entrenamiento, número de pares (n, C) utilizados para entrenar el codificador.
- *Tamaño de la ventana de selección*, ws, número de vecinos y propiedades considerados para construir los contextos de nodos de V.

En lo que sigue, notaremos por $\pi : V \to \mathbb{R}^D$ la proyección que hemos obtenido a partir del codificador neuronal entrenado.

Debido a que en la implementación y experimentos sobre bases de datos reales vamos a trabajar con grafos binarios con propiedades (no hipergrafos), tras haber obtenido una proyección sobre los nodos del grafo, se induce una proyección de las aristas en el mismo espacio vectorial (que notaremos también por π):

Definición 39. *Si $G = (V, E, \tau, \mu)$ es un Grafo con Propiedades, y $\pi : V \to \mathbb{R}^D$ es una inmersión de los nodos del grafo, damos una extensión de la inmersión al conjunto de aristas, $\pi : E \to \mathbb{R}^D$ de la siguiente forma:*

$$e \in E,\ s \xrightarrow{e} t,\ \textit{entonces}\ \pi(e) = \overrightarrow{\pi(s)\pi(t)}$$

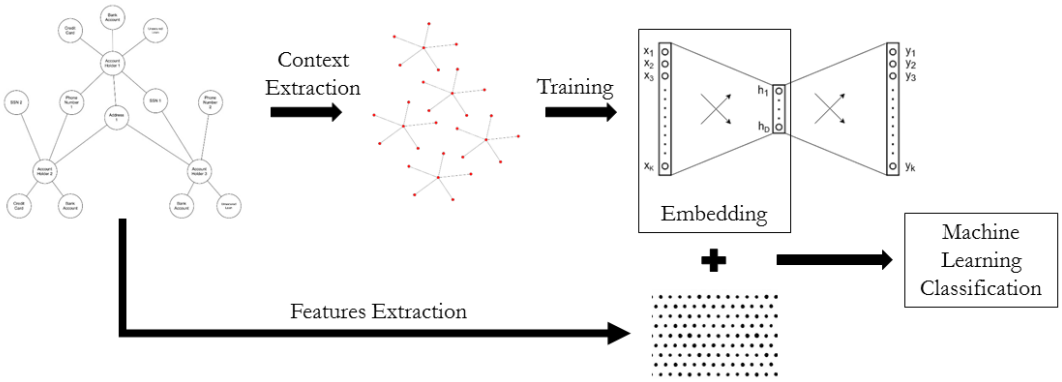


Figura 5.7: Representación esquemática de la metodología propuesta.

Como las operaciones habituales en espacios vectoriales son de uso extendido en las actuales unidades de cálculo (procesadores y GPUs), esta nueva representación puede ayudar a desarrollar algoritmos más eficientes para analizar, reparar y extraer información de conjuntos de datos multi-relacionales, en general, y más concretamente, de bases de datos en grafos. Por ejemplo, algunas tareas que pueden verse mejoradas con este tipo de inmersiones son:

- Clústers formados por nodos/aristas en el nuevo espacio vectorial pueden ayudar a asignar propiedades faltantes a los elementos de un grafo (haciendo uso de relaciones de distancia, linealidad o clusterización, por ejemplo).

- La representación vectorial de los elementos de un grafo puede ayudar a obtener medidas de similitud entre ellos.

- El análisis de los vectores asociados a las diferentes familias de relaciones (aquellas que comparten un tipo común, o verifican propiedades similares, por ejemplo) puede ayudar a detectar relaciones faltantes en el conjunto de datos original pero que en la nueva representación se hacen evidentes (la disposición de dos nodos cumple con el *vector representante* de algún tipo de relación, a pesar de que esa relación no aparece en el grafo).

- El análisis de la representación de las aristas que forman *caminos* en el grafo original puede ayudar a desarrollar formas más eficientes de detectar la existencia de dichos caminos en el grafo original.

5.4. Evaluación Empírica

Es momento ahora de realizar una evaluación empírica de nuestro método con dos objetivos claramente diferenciados:

1. analizar que las representaciones vectoriales que se obtienen a partir de grafos con propiedades mantienen características semánticas presentes en los mismos, y
2. evaluar diferentes aplicaciones que hacen uso de la inmersión propuesta para realizar tareas de clasificación y descubrimiento.

Para hacer estas comprobaciones, las diversas inmersiones que vamos a calcular no recibirán información acerca de los tipos de nodos o aristas que componen el grafo (formalmente, no recibirán información sobre la función τ). Los contextos asociados a los diferentes nodos del grafo, y que son utilizados para crear el conjunto de entrenamiento, son generados seleccionando aleatoriamente un número (*tamaño de la ventana de selección*) de nodos vecinos y de valores de sus diferentes propiedades en μ (sin τ).

Con el objetivo de verificar que determinadas estructuras *semánticas* presentes en la representación original $G = (V, E, \tau, \mu)$ se mantienen en la nueva representación del grafo analizaremos en qué medida la función τ, que determina los tipos de cada nodo o arista, puede ser deducida en la nueva representación a pesar de que el proceso de inmersión de G nunca recibió información sobre ella. Si es así, se verificaría que los tipos asociados a nodos y aristas pueden ser deducidos a partir de la estructura topológica original de la red y de las propiedades internas de sus elementos (V, E, μ) haciendo uso de algoritmos de aprendizaje.

5.4.1. Detalles de la implementación y experimentos

Entre las opciones barajadas, la arquitectura seleccionada para el codificador neuronal fue la arquitectura CBOW debido a que, a pesar de su simplicidad y el bajo coste computacional en su entrenamiento, obtiene buenos resultados en la tarea de capturar relaciones tanto *sintácticas* como *semánticas* entre los elementos codificados [151].

Como en el resto de aproximaciones que hemos realizado hasta el momento, se ha elegido Python 2.7 como lenguaje de programación para llevar a cabo la evaluación experimental señalada. Para la implementación de la arquitectura CBOW se ha utilizado el conjunto de herramientas *Gensim*[2] (versión 0.12.4), ya

[2]https://radimrehurek.com/gensim

que presenta grandes avances con respecto a las implementaciones anteriores de codificadores neuronales, consiguiendo mejoras como: posibilidad de analizar un corpus con tamaño mayor a la RAM disponible, API más intuitiva, fácil implantación y nuevas implementaciones (mejoradas y más escalables) de los algoritmos más populares de este tipo [191, 239]. Además, se ha utilizado Neo4j[3] (versión 2.2.5) como sistema de persistencia para los grafos con propiedades analizados. Para la comunicación con Neo4j desde Python se ha utilizado la librería py2neo[4] (versión 1.6.4).

Cada experimento de inmersión de los grafos con propiedades (en el espacio vectorial con los parámetros determinados) se ha repetido 10 veces, valor experimental que nos ha asegurado que la desviación estándar en los resultados experimentales con respecto a la predicción de los tipos de nodos y aristas haya sido inferior al 2 %. En el caso de las tareas relacionadas con *Entity Retrieval* estas desviaciones suben hasta el 7,8 %, y en el caso de la obtención de los nodos destino de un traversal han quedado acotadas por 8,9 %.

En lo relativo a las tareas de aprendizaje y modelos considerados para validar la inmersión se han utilizado: k-NN, Random Forest y Redes Neuronales. Para los test de clasificación generales, y salvo que se indique lo contrario, se ha utilizado k-NN con $k = 3$.

5.4.2. Datasets

Los experimentos se han llevado a cabo en 3 grafos con propiedades diferentes. Dos de ellos son ampliamente conocidos por la comunidad científica relacionada con el análisis de datos semánticos: WordNet y TheMovieDB. El tercero es un conjunto de datos desconocido para la comunidad denominado Ecuadorian Intangible Cultural Heritage.

Los conjuntos de datos han sido parcialmente manipulados para reducir su tamaño y complejidad por motivos de eficiencia. La Tabla 5.1 muestra algunas estadísticas sobre los conjuntos de datos tal y como han sido empleados en los experimentos.

A continuación damos algunos detalles acerca de cada uno de estos conjuntos para contextualizar las características que encontraremos en los resultados obtenidos.

WordNet® [79] es una base de datos de nombres, verbos, adjetivos y adverbios de la lengua inglesa. Es uno de los recursos más importantes en el área de lingüística computacional, y se ha construido como una combinación de dic-

[3] http://neo4j.com

[4] http://py2neo.org

Tabla 5.1: Estadísticas de los datasets

	Relation types	Node Types	Nodes	Relations
WordNet	12	4	97.539	240.485
TheMovieDB	4	5	66.020	125.624
EICH	10	11	38.990	55.358

Tabla 5.2: Tipos de aristas en WordNet

$also, domain\_category, domain\_member\_usage, domain\_region,$
$domain\_usage, hypernym, hyponym, instance\_hyponim$
$member\_meronym, part\_holonym, part\_meronym, member\_holonym.$

cionario y tesauro, ideado para que su uso sea intuitivo. Nació en el Laboratorio de Ciencias Cognitivas de la Universidad de Princeton gracias a la aportación de numerosos lingüistas, estudiantes e ingenieros de software, con el objetivo de dar soporte al análisis automático de textos y a diversas aplicaciones de Inteligencia Artificial, con un claro énfasis en aquellas que precisan del tratamiento semántico del lenguaje. Cada elemento en la base de datos representa lo que se ha llamado un *synset*: una lista de palabras sinónimas, y las relaciones que se establecen entre los elementos se dan tanto a nivel léxico como semántico, razón por la cual Wordnet ha sido ampliamente usada en el análisis sintáctico de textos y en entornos de extracción automática de información semántica.

Para este trabajo hemos utilizado una sección de la versión 3.0, considerando únicamente las entidades que están conectadas con relaciones pertenecientes a los tipos que se muestran en la Tabla 5.2 (de manera similar a [90]), obteniendo de esta forma un grafo con 97.593 nodos y 240.485 relaciones. Ejemplos de relaciones consideradas en WordNet son:

$$\texttt{correction} \xrightarrow{\texttt{hyponim}} \texttt{retribution}$$

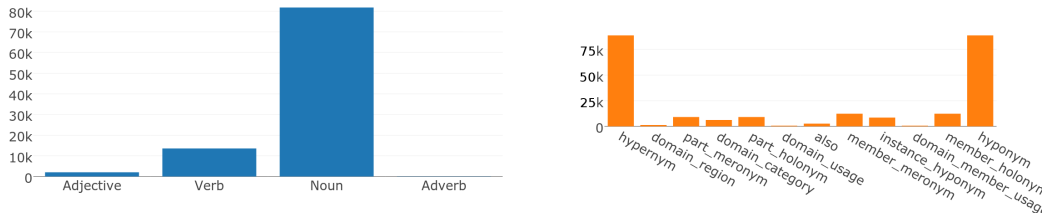
$$\texttt{spasm} \xrightarrow{\texttt{domain\_category}} \texttt{pathology}$$

En la Figura 5.8 mostramos la distribución por tipos de nodos y aristas en el grafo obtenido a partir de la sección de WordNet considerada, y la Figura 5.9 muestra una representación gráfica del esquema de datos presente en este dataset.

Por las características de la base de datos, las aristas no poseen propiedades, y los nodos presentan las siguientes propiedades:

- **id**: Identificador único del synset en la base de datos.

- **lexical_domain**: Dominio léxico al que pertenece el synset. Esta propiedad ha sido manipulada para nuestros experimentos, ya que originalmente contiene el tipo de synset (tipo de nodo) y es precisamente uno de nuestros objetivos de predicción para evaluar la corrección de la inmersión realizada. Ejemplos de cambios realizados son: `noun.substance` / `substance`, o `verb.perception` / `perception`.
- **label**: Etiqueta escrita en lenguaje natural definiendo el conjunto de palabras pertenecientes al synset. Ejemplos de esta propiedad son: `sexual_activity` o `scientific_method`.



(a) Distribución de nodos por tipo.

(b) Distribución de aristas por tipo.

Figura 5.8: Distribución de nodos y aristas en WordNet.

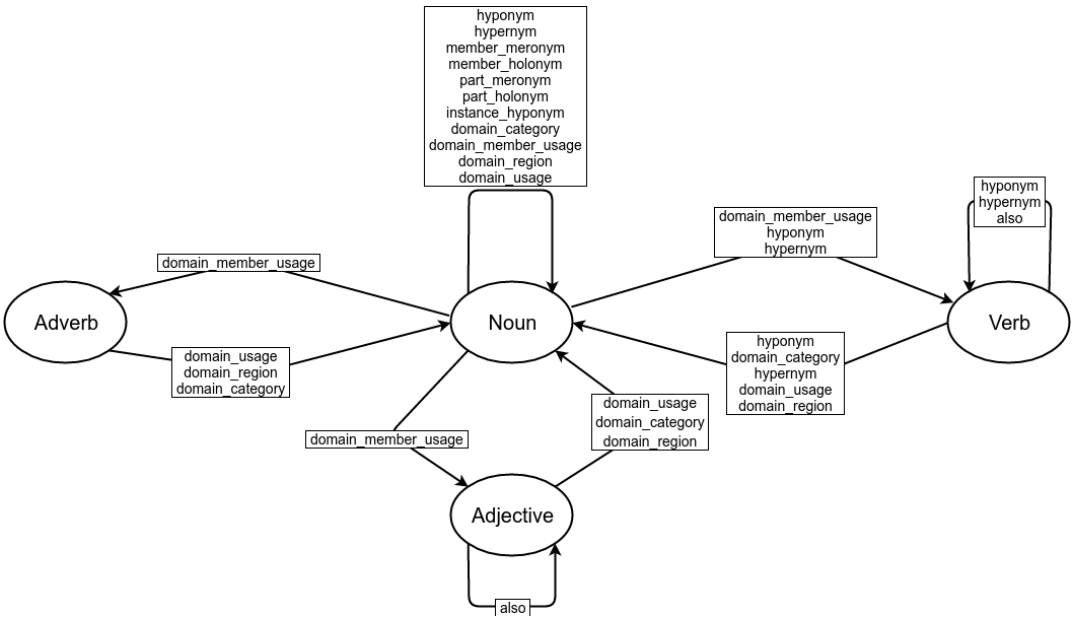


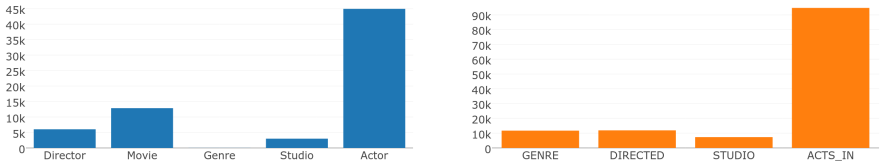
Figura 5.9: Esquema de datos de WordNet.

TheMovieDB (TMDb)[5] es un conjunto de datos que contiene información sobre actores, películas y contenidos de televisión. TMDb empezó en el año 2008 como un proyecto para ayudar a recopilar imágenes en alta resolución sobre películas, pero se ha convertido en una gran bases de datos sobre películas con una comunidad on-line altamente participativa y, en la actualidad, es la mayor base de datos de cine con acceso libre. Para nuestros experimentos hemos considerado todas las entidades de TMDb que están conectadas por relaciones pertenecientes a los tipos `acts_in, directed, genre` y `studio`, obteniendo un grafo con 66.020 nodos y 125.624 relaciones. Ejemplos de las relaciones presentes en TMDb son:

$$\texttt{steven\_spielberg} \overset{\texttt{directed}}{\rightarrow} \texttt{jurassic\_park}$$

$$\texttt{keanu\_reeves} \overset{\texttt{acts\_in}}{\rightarrow} \texttt{the\_matrix}$$

En la Figura 5.10 se muestra la distribución por tipos de nodos y aristas en el suconjunto de TMDb considerado y la Figura 5.11 muestra una representación gráfica del esquema de datos presente en este dataset.



(a) Distribución de nodos por tipo. (b) Distribución de aristas por tipo.

Figura 5.10: Distribución de nodos y aristas en TMDb.

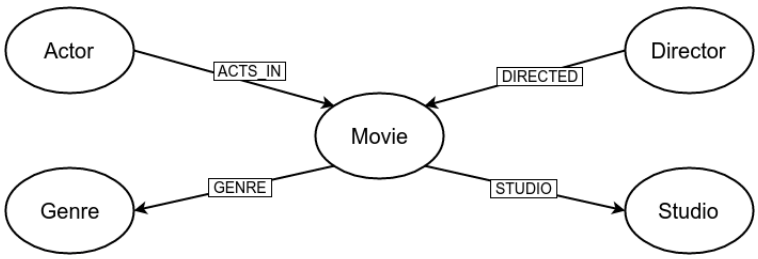


Figura 5.11: Esquema de datos de TMDb.

Cabe destacar que los tipos `Actor` y `Director` están solapados, concretamente, en nuestro conjunto existen 44.097 nodos que sólo tienen asignado el

[5]Disponible en https://www.themoviedb.org

tipo `Actor`, 5.191 nodos que sólo tienen asignado el tipo `Director`, y 846 nodos que poseen los tipos `Actor` y `Director` al mismo tiempo (nodos multi-tipo).

Las aristas en esta base de datos no poseen propiedades, y en los nodos se han considerado las siguientes propiedades:

- **name**: Propiedad presente en todos los nodos del grafo y que los identifica. Ejemplos de esta propiedad son `Keanu_Reeves`, o `The_Matrix`.
- **language**: Propiedad presente en los nodos de tipo `MOVIE`, indica la lengua original en la que la película fue rodada. Ejemplos de esta propiedad son `en` (inglés), o `es` (español).

Ecuadorian Intangible Cultural Heritage (EICH o Base de Datos del Patrimonio Cultural Inmaterial del Ecuador) corresponde a una sección de la base de datos del Instituto Nacional de Patrimonio Cultural Ecuatoriano [6] que contiene 38.990 nodos y 55.358 relaciones distribuidas a través de 11 tipos de nodos y 10 tipos de aristas, con información sobre el patrimonio cultural inmaterial del Ecuador. Esta base de datos es la más heterogénea de las 3 analizadas, presentando mayor tipología tanto en nodos como en aristas, y además sus elementos poseen más propiedades que las de los elementos de las otras dos bases consideradas. Ejemplos de relaciones en EICH son:

$$\texttt{quito} \xrightarrow{\texttt{localizacion}} \texttt{pichincha}$$

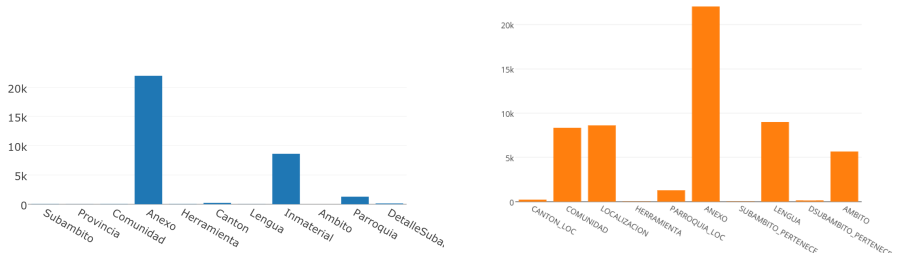
$$\texttt{canciones\_tradicionales\_pastaza} \xrightarrow{\texttt{lengua}} \texttt{achuar}$$

En la Figura 5.12 se muestra la distribución de nodos y aristas en EICH según sus tipos y en la Figura 5.13 mostramos una representación gráfica del esquema de datos presente en este dataset.

Las aristas en esta base de datos no poseen propiedades, y las propiedades consideradas en los nodos son:

- **Descripcion**: Está presente en todos los nodos y los identifica. Ejemplos de esta propiedad son `Pichincha`, o `Juegos_Rituales_o_Festivos`.
- **Denominacion2**: Presente en algunos nodos de tipo `Inmaterial`. Ofrece una denominación alternativa a la que se encuentra almacenada en `Descripción`.
- **Denominacion3**: Presente en algunos nodos de tipo `Inmaterial`. Ofrece una denominación alternativa a las que se encuentran almacenadas en `Descripción` y `Denominacion2`.

[6] Accesible desde http://www.inpc.gob.ec



(a) Distribución de nodos por tipo. (b) Distribución de aristas por tipo.

Figura 5.12: Distribución de nodos y aristas en EICH.

- **EntidadInvestigadora**: Presente en los nodos de tipo `Inmaterial`. Indica la entidad que anotó dicho patrimonio inmaterial en la base de datos.
- **FotoFotografo**: Presente en algunos nodos de tipo `Inmaterial`. Indica la persona que realizó la fotografía asociada a un patrimonio inmaterial, en caso de que exista.
- **AnioFoto**: Presente en algunos nodos de tipo `Inmaterial`. Indica el año en el que se tomó la fotografía asociada a un patrimonio inmaterial, en caso de que exista.
- **Tipo**: Presente en los nodos de tipo `Anexo`. Indica el tipo de anexo asociado a un patrimonio inmaterial, puede tomar los valores: `audio`, `archivo`, `imagen` y `video`.
- **Original**: Presente en los nodos de tipo `Anexo`. Indica el nombre del fichero asociado al que hace referencia.

Si medimos, de forma muy primitiva, la *riqueza semántica* de un nodo como:

$$rs(n) = \#(\mathcal{N}(n)) + \#(dom(\mu(n)))$$

es decir, la suma de la cantidad de relaciones en las que participa más la cantidad de propiedades que posee, podemos construir el histograma de riqueza semántica para cada uno de los conjuntos de datos anteriores (Figura 5.14). En el caso de WordNet, el promedio de riqueza semántica es $5,56$, en el caso de TMDb es $3,21$ y en el caso de EICH es $7,86$. Esta información puede ayudarnos a entender la composición de los casos estudiados y la interpretación de los resultados obtenidos.

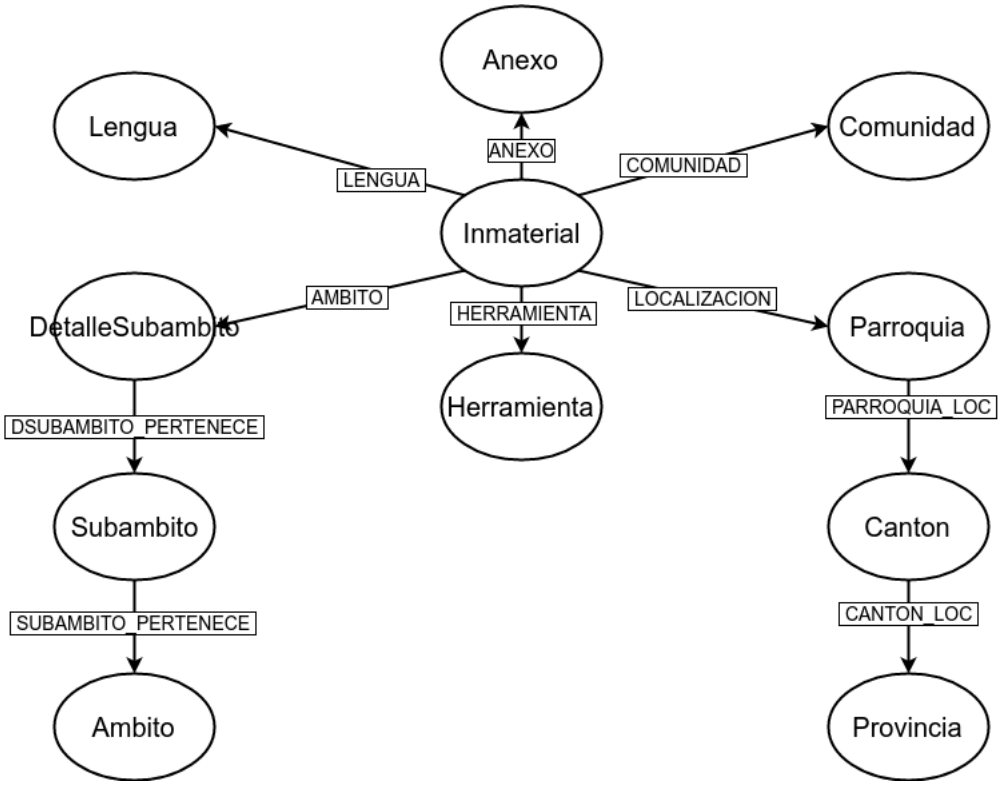
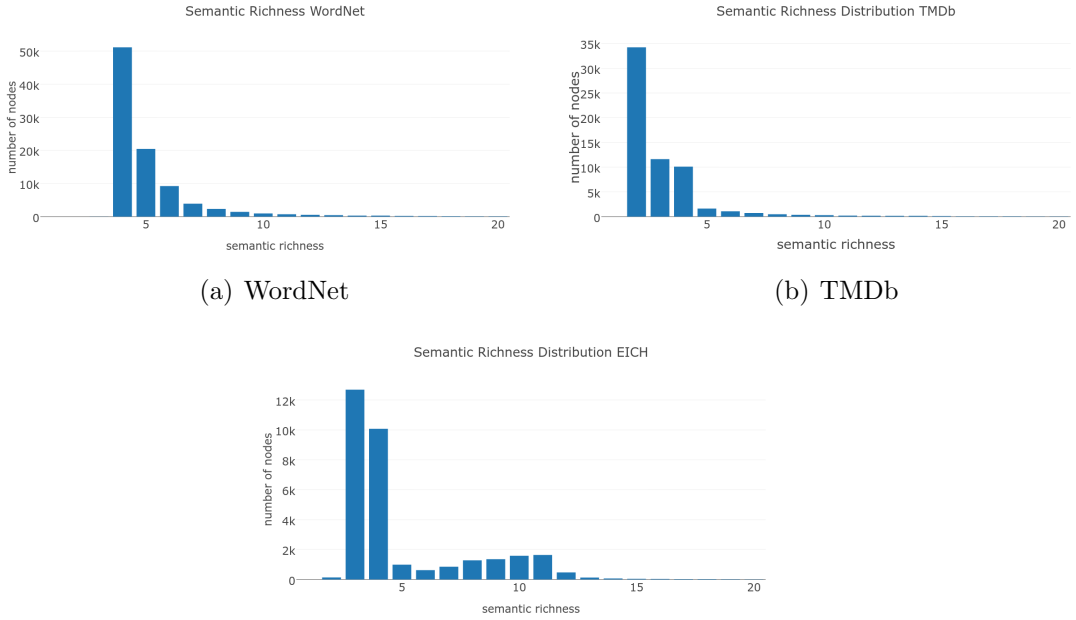


Figura 5.13: Esquema de datos de EICH.

Evaluación de la inmersión

Como se ha puesto de manifiesto, la inmersión que podemos hacer de un grafo con propiedades no es única, y no solo porque técnicamente es el resultado de una codificación neuronal que depende del proceso de entrenamiento completo, sino porque depende además de algunos hiperparámetros que intervienen en la ejecución del algoritmo, como son la dimensión del espacio de inmersión, el tamaño de la ventana utilizada para tomar los contextos de cada elemento, y el conjunto de entrenamiento (tanto del tamaño del conjunto, como de los elementos concretos que lo forman).

Ante esta situación, un primer problema que nos encontramos es determinar un proceso de evaluación que permita medir la bondad de las distintas inmersiones, indicando hasta qué punto son más o menos válidas. Tal y como mencionamos anteriormente, y teniendo en cuenta que uno de los objetivos deseados en las inmersiones es que mantengan las propiedades semánticas existentes en los datos originales, hemos decidido evaluar la capacidad de las nuevas representaciones para realizar tareas posteriores de clasificación automática haciendo uso de las nuevas representaciones obtenidas.



(a) WordNet

(b) TMDb

(c) EICH

Figura 5.14: Riqueza semántica (inferiores a 20).

5.4.3. Predicción de Tipos de Nodos

Como primera prueba intentaremos predecir la función τ que asocia tipos a los nodos del grafo y que, como comentamos anteriormente, no se proporciona durante el proceso de entrenamiento que calcula la inmersión. Por supuesto, de la misma manera que podemos intentar predecir la función τ, podríamos hacer una evaluación para predecir cualquier valor de la función μ que asigna diferentes propiedades a los elementos del grafo, pero teniendo la precaución de que la característica evaluada no haya sido utilizada durante el entrenamiento de inmersión.

Una primera intuición acerca de que las inmersiones conseguidas mantienen las estructuras semánticas (concretamente, el tipo de cada nodo) la podemos obtener analizando cómo se distribuyen los diversos tipos en el espacio vectorial sobre el que se ha hecho la inmersión.

La figura 5.15 muestra dos proyecciones de una sección de la inmersión obtenida para el grafo TMDb. La representación de la izquierda muestra una selección aleatoria de los vectores asociados a los nodos de tipo `Movie` y `Actor` haciendo uso de una inmersión en un espacio de dimensión 200 (que ha sido proyectada posteriormente sobre un espacio bidimensional haciendo uso de la técnica Multi-Dimensional Scaling [38] para facilitar su visualización), mientras

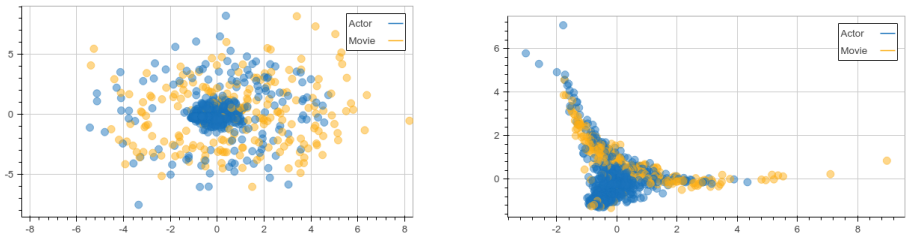


Figura 5.15: Representaciones 2D de nodos de tipo `Movie` y `Actor` en TMDb.

Tabla 5.3: Parámetros de inmersión

	N	D	**Tam. ventana**	**Predicción**
TMDb	400.000	150	3	$\simeq 72\,\%$
WordNet	1.000.000	50	8	$\simeq 96\,\%$
EICH	300.000	20	2	$\simeq 83\,\%$

que la representación de la derecha muestra la misma sección del grafo haciendo uso de una inmersión sobre un espacio de dimensión 2 directamente. A pesar de que la reducción de dimensionalidad considerada es a todas luces excesiva (pero necesaria para poder visualizar estos conjuntos en estas páginas), ambas representaciones muestran que las inmersiones de los nodos del grafo TMDb no se distribuyen aleatoriamente respecto del tipo, sino que siguen ciertos patrones, lo que evidencia que, en efecto, la inmersión obtenida mantiene información relativa al tipo de los nodos a pesar de que, como comentamos, la función τ nunca ha sido utilizada en su construcción.

Además de la libertad de elección en los parámetros que intervienen en la codificación, encontramos algunos grados de libertad adicionales al decidir qué máquina de aprendizaje se usará posteriormente para clasificar la inmersión de los nodos del grafo. Como primera aproximación, y a pesar de la alta carga computacional que demanda, se ha realizado un estudio exhaustivo de los parámetros libres del modelo haciendo uso del método de clasificación k-NN para explorar la dependencia que muestra la inmersión respecto del parámetro k. La razón por la que se ha elegido este modelo se centra en dos aspectos fundamentales: solo depende de un parámetro propio (el valor de k), que se sabe que funciona relativamente bien para $k = 3$ de forma general; y, a pesar de su simplicidad, proporciona resultados robustos que sirven de primer nivel de comparación para otros modelos de clasificación más sofisticados.

La Tabla 5.3 muestra valores de los parámetros en los que se han obtenido resultados de clasificación buenos usando k-NN como modelo posterior de

aprendizaje (con $k = 3$). La Figura 5.16 (al final de este capítulo) muestra los resultados de este análisis para los tres datasets considerados, donde se consiguen tasas de predicción superiores al 70 % para todos ellos.

En todos los casos se cumple que el tamaño del conjunto de entrenamiento, N, óptimo para realizar la predicción automática de los tipos de nodo es proporcional al número de nodos en el grafo. Tanto EICH como TMDb (para WordNet no conocemos el valor óptimo, porque es creciente en el rango analizado) muestran una reducción en la tasa de predicción a partir del valor óptimo, esto puede deberse a un sobreajuste relacionado con la existencia de nodos de diferentes tipos que poseen la misma etiqueta.

Respecto a la dimensión del espacio vectorial, se pueden observar leves cambios cuando aumentamos D por encima de 10-15, una dimensión relativamente baja, pero es casi imperceptible.

El estudio del parámetro *tamaño de la ventana de selección* muestra que se requieren valores pequeños de este parámetro para obtener buenos resultados en la predicción del tipo asociado a los nodos. Es importante señalar que la mejor predicción no se consigue en ningún caso con un *tamaño de la ventana de selección* igual a 1, ya que esto supondría que el sistema no necesita recibir pares nodo-contexto como elementos del conjunto de entrenamiento sino que bastaría con mostrarle instancias de las relaciones/propiedades presentes en cada nodo.

Comparación con otros modelos de predicción

Una vez fijados los parámetros de la inmersión que proporciona la Tabla 5.3, procedemos a comparar la capacidad predictiva con algunos métodos automáticos de clasificación sobre la misma tarea. Concretamente, compararemos estos resultados con los obtenidos a través de redes neuronales feedforward y Random Forest.

En la Figura 5.17 (también al final de este capítulo) se muestran los resultados obtenidos. La gráfica (a) muestra la variación de los resultados proporcionados por k-NN cuando varía el valor k; en (b) se muestran los resultados arrojados por Random Forest cuando se modifica el número de árboles; finalmente, en (c) se muestran los resultados de la red neuronal cuando se modifica el número de neuronas en la capa oculta. En cualquier caso, los valores de los parámetros de estos modelos se mantienen relativamente bajos para la tarea que se está llevando a cabo.

También presentamos las matrices de confusión promediadas tras realizar 10 experimentos utilizando los parámetros óptimos indicados anteriormente: WordNet (Tabla 5.4), TMDb (Tabla 5.5), y EICH (Tabla 5.9, al final del capítulo). Estas matrices capturan las similitudes semánticas entre los tipos de nodo. En

Tabla 5.4: Matriz de confusión: Predicción de tipos de nodos (WordNet)

	adjective	**verb**	**noun**	**adverb**
adjective	**90.01 %**	1.75 %	8.2 %	0.05 %
verb	0.44 %	**88.36 %**	11.19 %	0.0 %
noun	0.2 %	1.56 %	**98.23 %**	0.01 %
adverb	10.16 %	1.63 %	29.27 %	**58.94 %**

Tabla 5.5: Matriz de confusión: Predicción de tipos de nodos (TMDb)

	Director	**Movie**	**Genre**	**Studio**	**Actor**
Director	**11.45 %**	9.54 %	0.02 %	1.48 %	77.51 %
Movie	6.51 %	**65.8 %**	0.02 %	0.29 %	27.37 %
Genre	9.66 %	33.79 %	**2.07 %**	3.45 %	51.03 %
Studio	9.66 %	8.49 %	0.01 %	**1.23 %**	80.61 %
Actor	5.77 %	7.87 %	0.0 %	0.7 %	**85.66 %**

EICH, por ejemplo, `Canton`, `Parroquia` y `Provincia` muestran un comportamiento solapado debido a que todos ellos representan información geoespacial. En TMDb ocurre algo similar, `ACTOR` y `DIRECTOR` aparecen relacionados debido a que, como comentamos, existen numerosos nodos en esta base de datos que tienne ambos tipos.

5.4.4. Predicción de Tipos de Aristas

De forma similar al caso anterior, pasamos a realizar ahora un estudio orientado a determinar la bondad que presentan las inmersiones en la predicción de los tipos de aristas, que tampoco han sido usados en el proceso de entrenamiento del codificador neuronal.

La figura 5.18 muestra una proyección bidimensional (obtenida también aplicando el método MDS) de un conjunto de aristas seleccionadas aleatoriamente del conjunto de datos TMDb. Se puede observar que las aristas de tipo `Genre` no forman un único cluster, sino que está conformado por un conjunto de clústers periféricos que corresponden a los valores `Action, Comedy, Drama, Documentary, Horror` y `Crime`, mostrando un comportamiento semántico diferente al del resto de tipos. Esto podría indicarnos que `Genre` quizás no forma un tipo único desde el punto de vista semántico, y que su comportamiento refleja información acerca de cierta heterogeneidad que ha sido incluida en las decisiones de diseño al formar la base de datos original. Es aquí donde este

tipo de análisis muestra características que lo pueden hacer adecuado para ser usado como normalizador adicional a las bases de datos que cubre información semántica y no solo estructural.

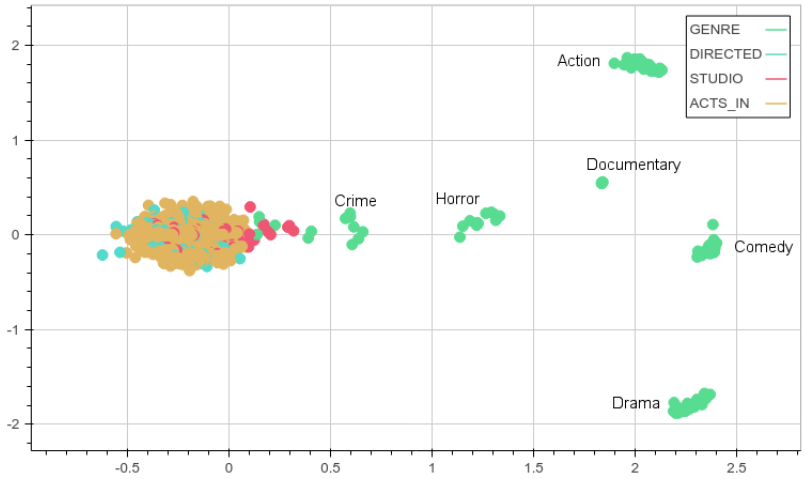


Figura 5.18: Representación 2D aristas de TMDb.

En la Figura 5.19 (también al final del capítulo) se muestran los resultados de la eficiencia de k-NN respecto a cambios en los parámetros de la inmersión. Teniendo en cuenta que el porcentaje de acierto está por encima del 80 % en todos los conjuntos de datos estudiados (incluso superando el 95 % en alguno de ellos), podemos concluir que la metodología seguida para la inmersión mantiene las propiedades semánticas también respecto de las aristas.

En general, podemos observar que el tamaño del conjunto de entrenamiento necesario para obtener buenos resultados a la hora de predecir los tipos de las aristas es superior al requerido para realizar una buena predicción de los tipos de nodo para los tres datasets analizados. Además, observando los resultados descubrimos que, a pesar de que WordNet es el grafo con propiedades que mejores resultados ofrecía en la predicción del tipo de los nodos, en el caso de la predicción de aristas se consiguen mejores resultados para EICH, logrando tasas con un valor $\simeq 97\,\%$ para los parámetros de la inmersión estudiados.

Al igual que en el caso anterior, las inmersiones requieren una dimensión relativamente baja, con pequeños cambios a partir de $D = 15$.

El comportamiento en la predicción de tipos de aristas según el *tamaño de la ventana de selección* muestra valores más elevados que los requeridos para la predicción en los tipos de nodos (superior a 20). En cualquier caso, es importante señalar que, de nuevo, la mejor predicción no se consigue en ningún caso con un *tamaño de la ventana de selección* igual a 1, lo que evidencia que muestrear los

Tabla 5.6: Matriz de confusión: Predicción de tipos de aristas (TMDb)

	GENRE	**DIRECTED**	**STUDIO**	**ACTS_IN**
GENRE	**99.51 %**	0.02 %	0.21 %	0.26 %
DIRECTED	0.01 %	**15.28 %**	2.04 %	82.67 %
STUDIO	0.13 %	7.22 %	**62.87 %**	29.79 %
ACTS_IN	0.01 %	4.75 %	0.94 %	**94.3 %**

contextos locales de un nodo consigue mejores resultados que el que se podría conseguir capturando sólo las relaciones binarias entre los nodos que ofrecen las aristas.

Comparación con otros modelos de predicción

Siguiendo la misma metodología que para tipos de nodos, en la Figura 5.20 (al final del capítulo también) se muestran los resultados obtenidos por los tres métodos de clasificación automática utilizados en el apartado anterior. Estas tareas de clasificación se han realizado con inmersiones que hacen uso de los parámetros presentados en la tabla 5.3, y se analizan modificando los mismos hiperparámetros de cada modelo concreto.

Las matrices de confusión tras promediar 10 experimentos para cada grafo se muestran en: TMDb (Tabla 5.6), EICH (Tabla 5.10), WordNet (Tabla 5.8), poniendo en evidencia que las inmersiones capturan similitudes entre diversos tipos de aristas. En el caso de EICH los tipos de aristas relacionados con información geo-espacial muestran un comportamiento solapado con las aristas de tipo `LENGUA`, debido a que existe una correlación entre las lenguas habladas y los territorios en los que se habla. WordNet muestra un comportamiento similar entre tipos de aristas `hypernym` y `hyponim`, debido a que tienen un comportamiento semántico similar. En el caso de TMDb, como era de esperar, las aristas de tipo `DIRECTED` se confunden con las aristas de tipo `ACTED_IN` debido al solapamiento entre los tipos de nodo `ACTOR` y `DIRECTOR` que intervienen en las mismas.

Los resultados experimentales correspondientes a la clasificación automática muestran que, para los conjuntos de datos analizados, la metodología propuesta conserva la semántica asociada a los tipos de aristas y muestra que la inmersión obtenida es capaz de detectar similitudes semánticas entre los tipos de aristas.

Cabe destacar que, debido a que pueden existir aristas de diferentes tipos entre el mismo par de nodos, el resultado en la predicción en el tipo de arista entre dos nodos puede haberse visto afectado. Si estamos tratando de predecir

el tipo de una arista entre dos nodos sólo a partir de las posiciones vectoriales de dichos nodos, y entre ellos existen aristas de diferentes tipos, la solución no es única, ya que cualquiera de estos tipos sería una solución válida. Este hecho no se ha tenido en cuenta a la hora de llevar a cabo los experimentos, por lo que los resultados en la predicción de tipos de aristas suponen un límite inferior y podrían ser mejorados teniendo en cuenta que la respuesta correcta no es única.

5.4.5. Entity Retrieval

Para seguir poniendo a prueba la bondad de las inmersiones que conseguimos respecto a la semántica interna de los grafos, y para poner de manifiesto su utilidad en otras tareas de predicción, vamos a evaluar hasta qué punto somos capaces de predecir *relaciones faltantes* haciendo uso de las inmersiones.

Para ello, consideraremos un subconjunto de aristas, $E' \in E$, que pertenecen al grafo original $G = (V, E, \tau, \mu)$ y que posteriormente eliminaremos, consiguiendo un subgrafo del anterior, $G' = (V, E \setminus E', \tau, \mu)$, sobre el que entrenaremos la inmersión. Posteriormente, trataremos de obtener el nodo destino asociado a cada arista en E' usando únicamente su nodo origen y $\pi(G')$.

Formalmente, dada una arista $e = (s, t) \in E'$, y dado el tipo de la relación, $\tau(e)$, que ha sido eliminada de manera previa a la inmersión del grafo, trataremos de obtener t a partir de $\pi(G')$, $\tau(e)$ y s. Esta tarea de obtener el target de una relación dado el nodo origen y el tipo de la misma es denominada habitualmente en la literatura como *Entity Retrieval* [48].

Vamos a utilizar el *vector representante* de los tipos para obtener el nodo destino de las relaciones faltantes, que definimos como:

Definición 40. *Dado un grafo con propiedades $G = (V, E, \tau, \mu)$, el* vector representante*, $\pi(\omega)$, asociado a un tipo de arista $\omega \in \tau(E)$, es el vector promedio de todos los vectores que representan a aristas de tipo ω.*

Si denotamos $E_\omega = \tau^{-1}(\{\omega\}) = \{e \in E : \ \tau(e) = \omega\}$, entonces:

$$\pi(\omega) = \frac{1}{\#(E_\omega)} \sum_{e \in E_\omega} \pi(e)$$

Si queremos obtener un candidato del destino de una relación e a partir del origen haciendo uso del *vector representante* de la relación y del vector asociado al nodo origen, basta hacer:

$$\pi(t_e) = \pi(s) + \pi(\tau(e))$$

Tabla 5.7: Ranking de Entity Retrieval usando la relación `hypernym`

	foam	*spasm*	*justification*	*neconservatism*
1	hydrazine	ejection	reading	pruritus
2	pasteboard	rescue	explanation	conservatism
3	silicon dioxide	putting to death	analysis	sight
4	humate	sexual activity	proposition	hawkishness
5	cellulose ester	behavior modification	religious doctrine	coma
6	synthetic substance	disturbance	accusation	scientific method
7	silver nitrate	mastectomy	assay	autocracy
8	cast iron	sales event	confession	judiciousness
9	sulfide	instruction	research	reverie
10	antihemorrhagic factor	debasement	discouragement	racism

El vector $\pi(t_e)$ representa la posición a la que *apunta* el *vector representante* de $\tau(e)$ desde el vector que representa el nodo origen $\pi(s)$ de la relación e. Una vez obtenido el vector $\pi(t_e)$ podemos obtener un *ranking* para los nodos del grafo, que se puede construir a partir de las distancias a $\pi(t_e)$ de cada vector asociado a los nodos del grafo original, de tal manera que los nodos que más cerca se encuentren del vector $\pi(t_e)$ serán posicionados en las primeras posiciones de dicho ranking.

En la tabla 5.7 se muestran los diez primeros resultados del ranking obtenido tras aplicar *Entity Retrieval* a través del vector representante, $\pi(hypernym)$, de las relaciones de tipo `hypernym` a diferentes nodos origen del grafo WordNet, los resultados están filtrados de tal manera que sólo se muestran los nodos de tipo `NOUN`.

Como nuestra metodología para construir la inmersión se realiza a partir de muestras aleatorias de diferentes *contextos locales* del grafo, es posible que en algunos casos el nodo origen de la relación no haya sido considerado en ningún momento y, por tanto, no podamos construir su representación vectorial. Para que este hecho no afecte a los resultados, en estos casos la arista no podrá ser evaluada y no influirá en el resultado experimental obtenido.

Para evaluar la bondad de la inmersión respecto de esta tarea haremos uso de la métrica *Mean Reciprocal Rank*, una métrica habitual en el área de *Information Retrieval*, y que ha sido utilizada en varios estudios de este tipo [48, 250].

Definición 41. *El* Reciprocal Rank *asociado a un resultado concreto, en una lista de posibles respuestas dada una consulta, es el inverso de la posición que ocupa ese resultado en dicha lista. El* Mean Reciprocal Rank *(MRR) es el pro-*

medio de los Reciprocal Ranks de todos los resultados en la lista asociada a una consulta determinada:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

donde Q representa el conjunto de consultas a evaluar y $rank_i$ la posición que ocupa en cada ranking la respuesta correcta.

En la Figura 5.21 se muestran los resultados obtenidos usando esta métrica sobre los datasets EICH, TMDb y WordNet en función del tamaño del conjunto de entrenamiento utilizado para realizar la inmersión, y eliminando del ranking aquellos nodos que no son del tipo que indica el nodo destino del tipo de relación evaluada. Como se puede observar en la gráfica, el método propuesto para llevar a cabo este tipo de tareas produce unos excelentes resultados que mejoran cuanto mayor es el conjunto de entrenamiento (en este caso, es un problema de multi-clasificación, por lo que no se pueden esperar resultados que se acerquen al 100 %).

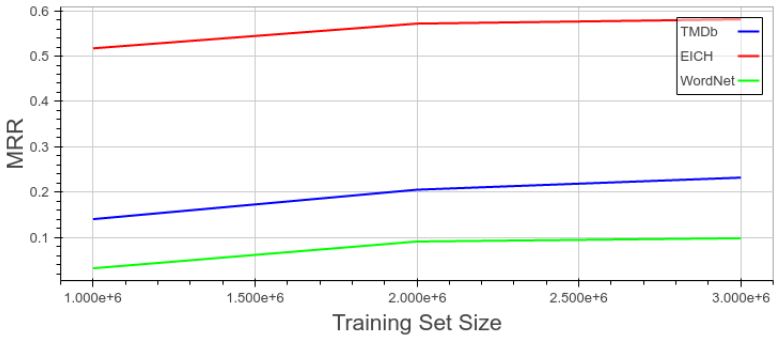


Figura 5.21: Análisis MRR en *Entity Retrieval*.

Este buen comportamiento nos permite obtener ciertas conclusiones sobre la estructura que los diferentes vectores (asociados a nodos y aristas) forman en la nueva representación: si el vector representante sirve para obtener el nodo destino de una arista significa que existe poca desviación entre las aristas del mismo tipo. Por otro lado, los nodos origen y destino del tipo de arista que se usa deben estar lo suficientemente dispersos para que, utilizando el vector representante, se consigan buenos resultados en cuanto a tareas relacionadas con *Entity Retrieval*.

5.4.6. Inmersión de caminos tipados

Por último, y solo a modo de demostración de las posibilidades que abre el tener una correcta representación vectorial de los elementos de un grafo con propiedades, presentamos una técnica basada en la inmersión para obtener el nodo destino de un *camino tipado* dado el tipo del camino y el nodo origen del mismo.

Un camino tipado no es más que la sucesión de tipos en nodos y aristas que corresponde a un camino dentro del grafo (en algunos contextos a estos caminos tipados se les conoce como *traversals*, pero preferimos no usar esta nomenclatura debido al solapamiento que produce con los traversals como métodos de consulta en determinados lenguajes de consulta sobre grafos). Formalmente:

Definición 42. *Un* camino tipado *de un grafo con propiedades* $G = (V, E, \tau, \mu)$ *es una sucesión*

$$T = t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} \ldots \xrightarrow{r_q} t_{q+1}$$

donde $t_i \in \tau(V)$ *(es un tipo válido para los nodos) y* $r_i \in \tau(E)$ *(es un tipo válido para las aristas). Denotaremos por* $Tp(G)$ *el conjunto de posibles caminos tipados de* G*.*

Definición 43. *En las condiciones anteriores, podemos definir la aplicación* Tp *que asocia a cada posible camino tipado de* G *el conjunto de caminos que verifican el patrón de tipos especificado por él:*

$$Tp : Tp(G) \to \mathcal{P}(G)$$

tal que si $T = t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} \ldots \xrightarrow{r_q} t_{q+1}$*, entonces para todo* $\rho \in Tp(T)$ *se verifica que:*

1. $\tau(sop_V(\rho)) = (t_1, \ldots, t_{q+1})$.
2. $\tau(sop_E(\rho)) = (r_1 \ldots, r_q)$.

Nuestro objetivo es obtener el nodo destino de un camino existente dado el nodo origen del mismo y el camino tipado que verifica. En este caso no eliminamos los tipos antes de realizar la inmersión, pues no tratamos de hacer predicción sino de ofrecer un nuevo mecanismo para la obtención del nodo destino de un camino que permita mejorar los tiempos que requieren este tipo de consultas, ya que en los sistemas tradicionales tienen un coste computacional muy elevado.

Para ello, definimos el vector representante de un camino de forma similar a como lo hicimos en la tarea anterior (que realmente se puede considerar un caso particular de camino tipado para caminos de longitud 1).

Definición 44. *El* vector representante *de un camino, $n_1 \overset{\rho}{\rightsquigarrow} n_2$, en un grafo con propiedades, es el vector que separa la representación vectorial del nodo origen del camino, $\pi(n_1)$, y la representación vectorial del nodo destino del mismo, $\pi(n_k)$. Es decir:*

$$\pi(\rho) = \overrightarrow{\pi(n_1)\pi(n_k)} = \pi(n_k) - \pi(n_1)$$

El vector representante *asociado a un camino tipado, T, es el vector promedio de todos los vectores que verifican el patrón de tipos especificado por T, es decir:*

$$\pi(T) = \frac{1}{|Tp(T)|} \sum_{\rho \in Tp(T)} \pi(\rho)$$

Como ocurría con los tipos de aristas, es posible que en algunos casos el nodo origen del camino no haya sido tomado en la muestra de la inmersión y, por tanto, no exista su representación vectorial. En estos casos, dicho camino no podrá ser evaluado y no influirá en el resultado experimental obtenido.

A partir de esta definición se han realizado experimentos para evaluar la tarea de obtener el nodo destino de un camino dado el nodo origen y el vector representante del camino tipado asociado. Para ello, hemos filtrado los nodos destino según el tipo indicado por el último elemento de la secuencia que define el camino tipado y hemos utilizando de nuevo la métrica MRR presentada en el apartado anterior. Los experimentos han sido realizados sobre el dataset EICH debido a que presenta una estructura más compleja en sus tipos que el resto de datasets y permite la construcción de caminos tipados más complejos.

En la figura 5.22 se muestran los resultados obtenidos en los experimentos haciendo uso de los siguientes caminos tipados (los tipos de los nodos se representan en minúsculas, y los de las aristas se omiten porque representan el único tipo de arista que permite el esquema mostrado en la figura 5.13):

1. $T1 = (Inmaterial \overset{r_1}{\rightarrow} DetSubambito \overset{r_2}{\rightarrow} Subambito \overset{r_3}{\rightarrow} Ambito)$

   Está asociado a caminos de longitud 3 y contiene información sobre a qué Ambito (existen 5 ámbitos diferentes en EICH) pertenece cada elemento del patrimonio inmaterial almacenado en el grafo.

2. $T2 = (Inmaterial \overset{r_1}{\rightarrow} Parroquia \overset{r_2}{\rightarrow} Canton \overset{r_3}{\rightarrow} Provincia)$

   Está asociado a caminos de longitud 3 y contiene información sobre a qué Provincia (existen 24 provincias diferentes en EICH) pertenece cada elemento del patrimonio inmaterial almacenado en el grafo.

Figura 5.22: Análisis MRR en caminos tipados.

En los resultados se aprecia el buen desempeño de la tarea propuesta para caminos tipados de tipo $T1$, cercana al 70 %, para un tamaño del conjunto de entrenamiento de tamaño superior a 3 millones, (o superior). En el caso del camino tipado $T2$, sin embargo, su resultado tiende a empeorar cuando aumentamos el tamaño del conjunto de entrenamiento por encima de 1 millón, llegando a estar por debajo del 25 %. En cualquier caso, el problema asociado a T_2 es considerablemente más complejo, ya que hay más de 20 provincias, frente a los 5 posibles ámbitos para el primer caso.

Aunque, por supuesto, harían falta más pruebas para validar la validez de esta metodología, esta aplicación muestra que un sistema como éste podría utilizarse para aproximar el resultado de consultas *a larga distancia* en bases de datos, que son especialmente ineficientes en el caso de los sistemas clásicos de persistencia, por lo que la inmersión se presenta como una alternativa interesante que, aunque reduciendo la fiabilidad del resultado, permite agilizar enormemente la carga computacional requerida.

Tabla 5.8: Matriz de confusión: Predicción de tipos de aristas (WordNet)

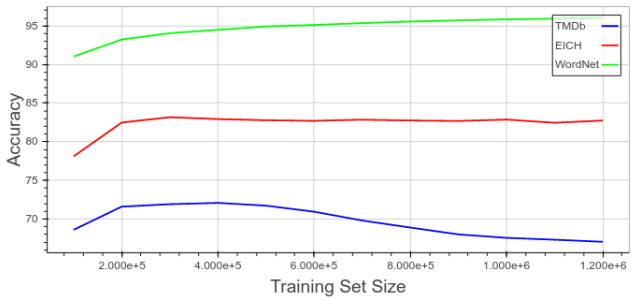
	hyper	dom_reg	part_mero	dom_cat	part_holo	dom_usage	also	memb_mero	inst_hypo	dom_memb_usage	memb_holo	hypo
hyper	**83.67 %**	0.01 %	1.14 %	0.76 %	0.43 %	0.02 %	0.56 %	1.96 %	0.13 %	0.02 %	2.77 %	8.52 %
dom_reg	2.08 %	**65.58 %**	25.0 %	0.33 %	0.93 %	0.0 %	0.21 %	3.27 %	0.24 %	0.0 %	1.12 %	1.25 %
part_mero	29.21 %	0.74 %	**44.47 %**	1.1 %	3.54 %	0.01 %	0.79 %	4.12 %	0.34 %	0.0 %	4.47 %	11.2 %
dom_cat	14.88 %	0.02 %	1.19 %	**78.91 %**	0.11 %	0.01 %	0.44 %	0.18 %	0.04 %	0.0 %	0.25 %	3.97 %
part_holo	15.53 %	0.09 %	3.28 %	0.34 %	**45.36 %**	0.01 %	0.78 %	2.25 %	1.24 %	0.0 %	6.67 %	24.45 %
dom_usage	4.28 %	0.0 %	0.06 %	0.39 %	0.03 %	**93.41 %**	0.99 %	0.06 %	0.0 %	0.0 %	0.11 %	0.68 %
also	8.0 %	0.0 %	0.38 %	0.43 %	0.19 %	0.03 %	**78.0 %**	1.92 %	0.07 %	0.01 %	4.64 %	6.35 %
memb_mero	11.7 %	0.18 %	1.76 %	0.2 %	0.77 %	0.0 %	0.93 %	**50.72 %**	0.21 %	0.0 %	24.67 %	8.87 %
inst_hypo	2.47 %	0.06 %	0.87 %	0.05 %	1.81 %	0.0 %	0.2 %	0.53 %	**80.42 %**	0.0 %	1.56 %	12.02 %
dom_memb_usage	1.1 %	0.0 %	0.03 %	0.03 %	0.14 %	0.0 %	1.13 %	0.09 %	0.06 %	**93.62 %**	0.09 %	3.73 %
memb_holo	11.24 %	0.05 %	0.67 %	0.09 %	1.83 %	0.0 %	0.91 %	14.01 %	0.21 %	0.0 %	**62.37 %**	8.61 %
hypo	10.52 %	0.01 %	0.42 %	0.31 %	1.18 %	0.02 %	0.57 %	1.23 %	1.18 %	0.01 %	3.67 %	**80.88 %**

Tabla 5.9: Matriz de confusión: Predicción de tipos de nodos (EICH)

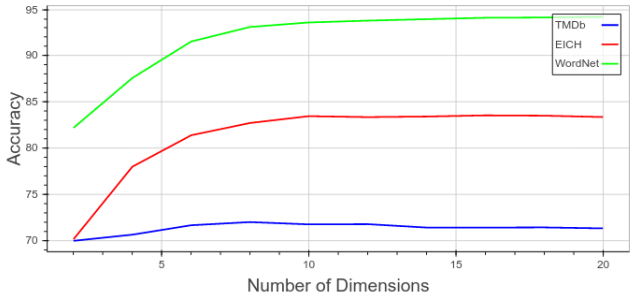
	Subambito	Provincia	Comunidad	Anexos	Herramienta	Canton	Lengua	Inmaterial	Ambito	Parroquia	DetalleSubambito
Subambito	**14.53 %**	0.0 %	0.0 %	2.56 %	0.0 %	0.0 %	0.0 %	47.86 %	0.0 %	0.85 %	34.19 %
Provincia	0.0 %	**7.14 %**	2.04 %	0.0 %	0.0 %	69.39 %	5.1 %	1.02 %	0.0 %	15.31 %	0.0 %
Comunidad	0.0 %	0.0 %	**0.0 %**	7.91 %	0.0 %	1.44 %	0.0 %	25.18 %	0.0 %	65.47 %	0.0 %
Anexos	0.0 %	0.0 %	0.0 %	**81.16 %**	0.0 %	0.0 %	0.0 %	18.63 %	0.0 %	0.21 %	0.0 %
Herramienta	0.0 %	0.0 %	0.0 %	0.68 %	**36.99 %**	0.0 %	0.0 %	62.33 %	0.0 %	0.0 %	0.0 %
Canton	0.0 %	3.74 %	0.1 %	5.27 %	0.0 %	**12.18 %**	0.0 %	24.26 %	0.0 %	54.27 %	0.19 %
Lengua	0.0 %	0.0 %	0.0 %	6.25 %	0.0 %	0.0 %	**0.0 %**	21.25 %	0.0 %	72.5 %	0.0 %
Inmaterial	0.01 %	0.0 %	0.0 %	9.44 %	0.19 %	0.0 %	0.01 %	**89.77 %**	0.0 %	0.56 %	0.01 %
Ambito	44.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	28.0 %	**0.0 %**	0.0 %	28.0 %
Parroquia	0.02 %	0.42 %	0.08 %	2.34 %	0.02 %	2.11 %	0.35 %	29.67 %	0.0 %	**64.82 %**	0.18 %
DetalleSubambito	1.63 %	0.0 %	0.0 %	5.42 %	0.0 %	0.18 %	0.0 %	49.37 %	0.0 %	4.52 %	**38.88 %**

Tabla 5.10: Matriz de confusión: Predicción de tipos de aristas (EICH)

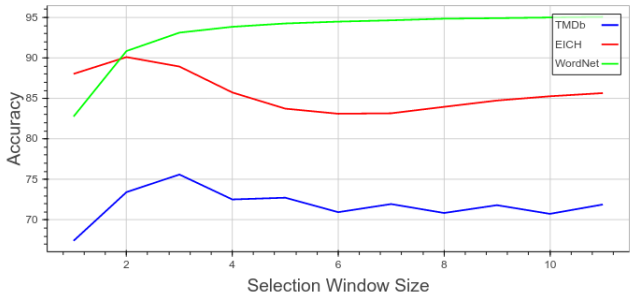
	CANTON_L	COM	LOC	HERRAM	PARROQ_L	ANEXO	SUBAMBITO_P	LENGUA	AMBITO	DSUBAMBITO_P
CANTON_L	**25.05 %**	1.94 %	12.62 %	0.0 %	26.8 %	6.99 %	0.0 %	22.91 %	3.69 %	0.0 %
COM	0.0 %	**97.92 %**	0.18 %	0.0 %	0.09 %	0.37 %	0.0 %	1.4 %	0.04 %	0.0 %
LOC	0.0 %	0.12 %	**96.77 %**	0.04 %	1.08 %	1.51 %	0.0 %	0.33 %	0.15 %	0.0 %
HERRAM	0.0 %	0.0 %	1.49 %	**44.03 %**	2.24 %	48.51 %	0.0 %	3.73 %	0.0 %	0.0 %
PARROQ_L	0.89 %	0.99 %	13.95 %	0.02 %	**59.11 %**	3.34 %	0.0 %	19.2 %	2.45 %	0.05 %
ANEXO	0.14 %	0.14 %	0.49 %	0.02 %	2.46 %	**95.87 %**	0.0 %	0.73 %	0.11 %	0.05 %
SUBAMBITO_P	0.81 %	0.0 %	10.57 %	0.0 %	8.13 %	9.76 %	**30.89 %**	14.63 %	5.69 %	19.51 %
LENGUA	0.0 %	1.06 %	0.09 %	0.0 %	0.01 %	0.39 %	0.0 %	**98.34 %**	0.1 %	0.0 %
AMBITO	0.01 %	0.04 %	0.28 %	0.01 %	0.04 %	1.9 %	0.0 %	2.38 %	**95.33 %**	0.02 %
DSUBAMBITO_P	0.18 %	0.18 %	1.96 %	0.0 %	2.67 %	3.21 %	0.89 %	22.46 %	7.66 %	**60.78 %**



(a) En función del tamaño del conjunto de entrenamiento.

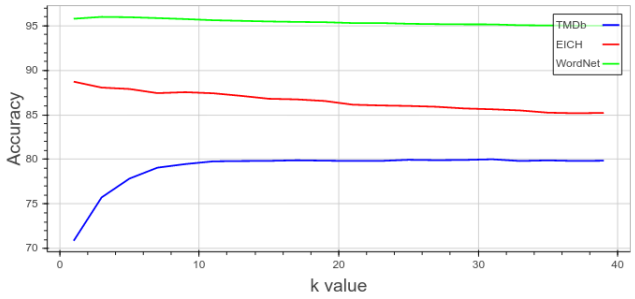


(b) En función del número de dimensiones.

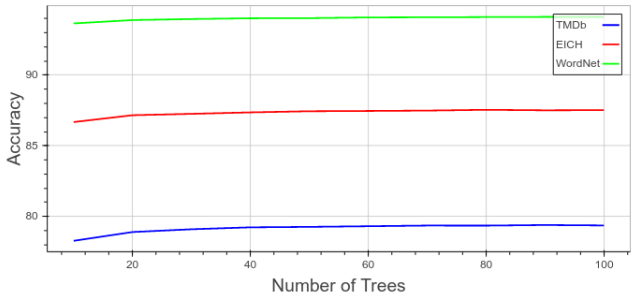


(c) En función del *tamaño de la ventana de selección.*

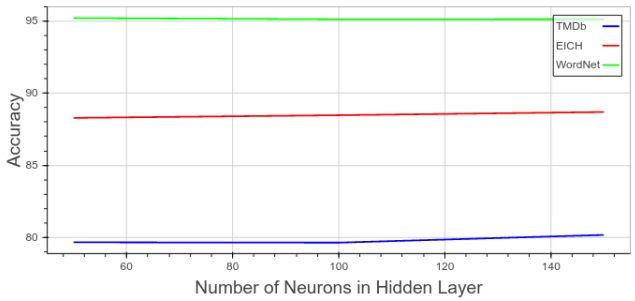
Figura 5.16: Análisis de la inmersión (predicción de tipos de nodo).



(a) k-Nearest Neighbor.

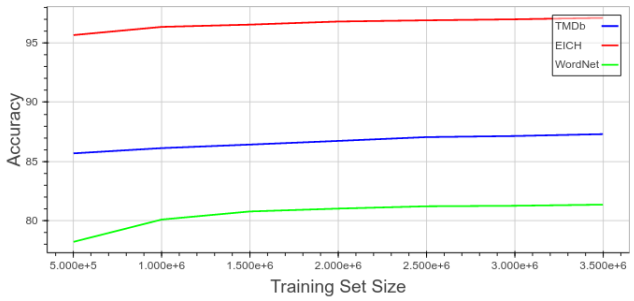


(b) Random Forest.

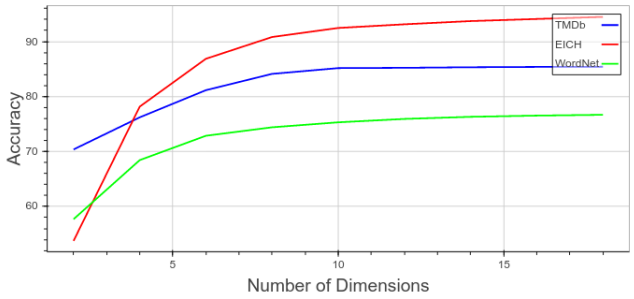


(c) Red Neuronal Artificial.

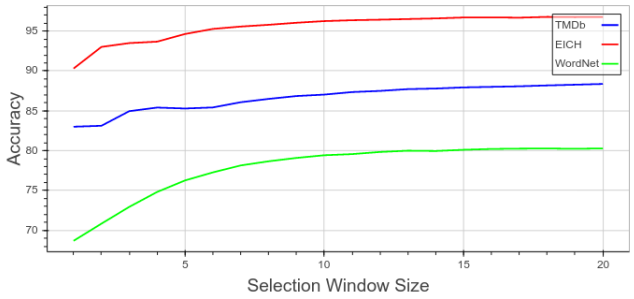
Figura 5.17: Análisis clasificación tipos de nodo por diferentes métodos.



(a) En función del tamaño del conjunto de entrenamiento.

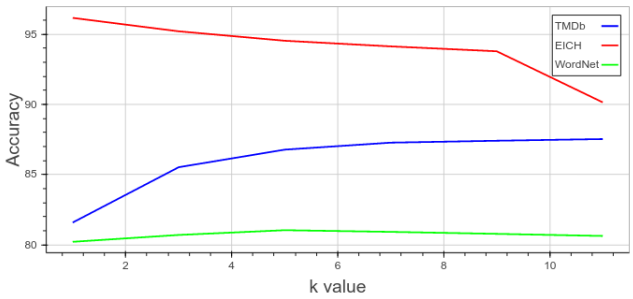


(b) En función del número de dimensiones.

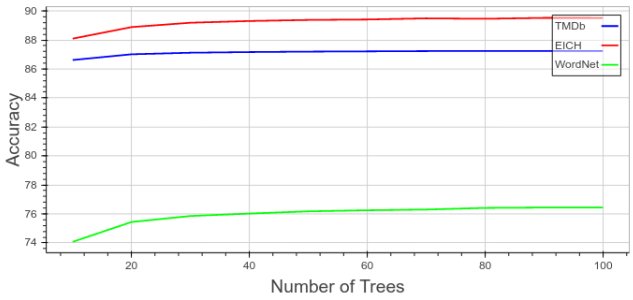


(c) En función del *tamaño de la ventana de selección.*

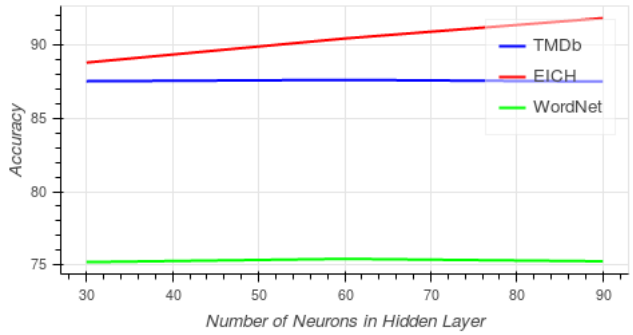
Figura 5.19: Análisis de la inmersión (predicción de tipos de arista).



(a) k-Nearest Neighbor.



(b) Random Forest.



(c) Red Neuronal Artificial.

Figura 5.20: Análisis clasificación tipos de arista por diferentes métodos.

Capítulo 6

Conclusiones y Trabajo Futuro

Como se ha comentado en diversos apartados de esta memoria, y a pesar de su potencial, el aprendizaje automático que hace uso de información relacional ha estado en un segundo plano en relación al aprendizaje automático más estándar, que hace uso de información no relacional, habitualmente en forma de tablas y otras estructuras más regulares. Tras el trabajo de estudio realizado, podemos identificar ciertos motivos que han llevado a que se produzca esta situación, algunos de ellos, por supuesto, no se presentan únicamente en este contexto.

Por una parte, hemos detectado que la comunidad científica posee cierta inercia en su investigación, lo que le lleva a priorizar la optimización y modificación de los algoritmos existentes por encima de la creación de nuevos algoritmos, o el uso de nuevas estructuras de datos, a partir de los cuales realizar el aprendizaje. En cierto sentido, determinadas publicaciones científicas obtienen impacto superando resultados anteriores bien fundados y, sin duda, este puede ser un motivo por el cual adentrarse en nuevos caminos no es tan fructífero (curricularmente hablando) para el investigador como continuar por una senda ya trazada, pues esta última opción habitualmente no encuentra las plataformas adecuadas para la presentación de su trabajo o, simplemente, exige un mayor esfuerzo de validación si plantea una metodología nueva que si mejora una metodología que ya ha demostrado estar bien fundada y ser útil. Además, las nuevas vías suelen ir acompañadas de un tiempo inicial de escasez de resultados publicables.

Por otro lado, las bases de datos que más habitualmente se han utilizado, y en las que se encuentra almacenada la información referente a la mayoría de

los fenómenos estudiados, hacen uso de esquemas y sistemas basados en bases de datos relacionales. Como ponen de manifiesto muchos estudios realizados, estas bases de datos clásicas no muestran un desempeño óptimo al trabajar con relaciones complejas, lo que supone una razón más para el impedimento de que se desarrollen metodologías orientadas a ponderar de igual forma los elementos almacenados y sus relaciones.

Adicionalmente, la mayor riqueza expresiva de las estructuras de representación de la información más complejas impone una mayor dificultad a la hora de realizar nuevos algoritmos y proporciona, al menos en las primeras aproximaciones, resultados menos llamativos que los métodos más depurados y más tradicionales.

Por último, se produce un proceso de retroalimentación del tipo *rich gets richer* (*el dinero llama al dinero*), en el que los métodos más convencionales son más conocidos por la comunidad científica y, por tanto, llegan a más investigadores porque son más distribuidos, lo que provoca que la mayoría de los investigadores que trabajan en Aprendizaje Automático acaban conociendo, principalmente, las metodologías más clásicas de almacenamiento y aprendizaje.

Nos gustaría enfatizar nuestra creencia en que razonar sobre las estructuras formales utilizadas para almacenar la información referente a los sistemas que analizamos es esencial si se quiere llevar a cabo este tipo de tareas de manera óptima y extraer el mayor beneficio de los datos adquiridos. Muchas veces, los investigadores transforman los datos obtenidos al medir un sistema a alguno de los formatos regulares habituales, normalmente vectores o tablas, perdiendo con frecuencia facetas de información relacional importante cuya expresión no es natural en estos formatos. A la hora de analizar un sistema a través de técnicas derivadas del Aprendizaje Automático puede llegar a ser tan importante la estructura de datos utilizada para expresar la información como el propio modelo de aprendizaje utilizado. Tras el estudio realizado para llevar a cabo esta investigación, consideramos que no existe un esfuerzo proporcional en el área entre optimizar las estructuras de las que se aprende y los métodos de aprendizaje elegidos. Paralelamente, y como ya se ha comentado, se debe realizar un esfuerzo por crear modelos de aprendizaje que sean capaces de aprender de estructuras más allá de las comunes y regulares.

En este trabajo hemos explorado la capacidad de aprendizaje a partir de

datos estructurados en forma de grafos con propiedades pero existen otras estructuras que, con seguridad, merece la pena ser consideradas. Por ejemplo, la información adquirida de los fenómenos dinámicos, normalmente almacenados en forma de series temporales, no han gozado tampoco del mismo nivel de atención, posiblemente debido a la mayor complejidad que presentan, aunque en los últimos años se ven indicios de un mayor interés por parte de la comunidad científica que las aborda por medio de herramientas de aprendizaje automático. Esta es la razón principal por la que se ha considerado de interés introducir en el capítulo 2 una estructura matemática más general que las tradicionales y que proporciona el soporte para poder desarrollar modelos de aprendizaje complejos sobre datos relacionales.

Con respecto al aprendizaje en grafos con propiedades, cabe destacar que existen varias líneas de trabajo que transforman los datos originales (en forma de grafo) hacia otras estructuras que los algoritmos más extendidos son capaces de manejar de manera más natural (debido a que fueron creados para trabajar específicamente con dichas estructuras). Es el caso de las inmersiones de grafos en espacios vectoriales [178, 226, 95], así como de la propuesta presentada en el apartado 5.3, que muestrean el grafo a partir de subestructuras para acomodarlo a una colección de objetos (pares $(elemento, contexto)$) que los algoritmos tradicionales pueden consumir de manera óptima. También es el caso de los trabajos que usan Redes Neuronales Convolucionales para realizar tareas de aprendizaje en grafos [125, 65, 69], que suelen ser habituales en procesamiento de imágenes. Una aproximación similar se ha seguido en trabajos relacionados con el análisis de series temporales [242], transformando éstas a una estructura que pueda ser tratada de manera natural por una red neuronal convolucional. Para poder utilizar este tipo de modelos sobre grafos debemos definir qué se entiende por el contexto espacial de un elemento del grafo, haciendo suposiciones que incluyen un sesgo adicional a la información analizada. Desde nuestro punto de vista, éstas son aproximaciones válidas que deben seguir siendo investigadas, pero se deben considerar otras opciones como la de trabajar directamente con la estructura de grafo, que ha sido una de las líneas de investigación seguidas en este trabajo y que ha demostrado ser válida.

En ocasiones, los esfuerzos al trabajar con datos se centran en conseguir predicciones automáticas que mejoren cuantitativamente resultados anteriores,

normalmente medidos a través de *benchmarks* que son utilizados como herramientas evaluadoras. Sin embargo, hay métodos relacionados con la predicción que pueden aportar resultados (cuantitativos y cualitativos) que no son fácilmente medibles a través de este tipo de técnicas. Además, existen otras tareas interesantes, y no relacionadas con la predicción, que se pueden llevar a cabo con datos y que no han recibido la atención que merecen en el área. Por ejemplo, los análisis relacionados con la *pureza semántica* de un conjunto de datos pueden ser interesantes a la hora de evaluar la estructura de los mismos y detectar incongruencias tales como el solapamiento entre tipos de datos o la presencia de redundancia en los datos o en el esquema, así como mejorar la eficiencia de consultas sobre estos conjuntos de datos.

Otra tarea relacionada, pero que consideramos que no recibe la atención que merece, es el descubrimiento de información a través de modelos de caja blanca. Si analizamos la tendencia en el aprendizaje, relacional o no, podemos observar que en los últimos años se han invertido muchos esfuerzos en métodos de caja negra, quedando en un segundo plano los métodos explicativos. Del lado de modelos de caja blanca en el área relacional, algoritmos explicativos como MRDTL surgieron al final de la década de los 90 y parece que su desarrollo se ha estancado en los últimos años. Sin duda, los modelos de caja negra están mostrando resultados impactantes, e inesperados hace pocos años, por medio de su máximo exponente, el Deep Learning, pero cuya interpretación es demasiado difusa para ser comprendida por un humano (aunque hay muchos esfuerzos por crear herramientas adicionales que puedan disminuir esta brecha). Consideramos que este hecho puede ser peligroso ya que, a pesar de que los métodos de caja negra nos permiten predecir hasta cierto punto la evolución de los sistemas, al prescindir de una explicación que pueda ser entendida por un humano evita que éste realice un aprendizaje real sobre cómo funciona el sistema, limitándose a proporcionar una herramienta que lo predice pero que no añade un conocimiento adicional al investigador. De esta forma se convierten en herramientas útiles para la Ingeniería (lo que justifica, por supuesto, el trabajo realizado sobre ellos), pero en menor medida para la Ciencia menos aplicada. Si queremos conocer los fenómenos que nos rodean y avanzar en el entendimiento de nuestro entorno no podemos conformarnos con crear máquinas que lo predigan, sino que debemos buscar formas de comprenderlo. Por todo ello, consideramos que se debe realizar un esfuerzo superior en el estudio y desarrollo de métodos de

aprendizaje automático de caja blanca.

A continuación haremos un recorrido más detallado sobre algunas conclusiones que se pueden derivar de las diversas aproximaciones que conforman este trabajo.

6.1. De los PQG y su uso para PQG-ID3

En el capítulo 3 se abordó el objetivo de obtener una herramienta para evaluar subgrafos inmersos en grafos con propiedades que pueda ser utilizada en procedimientos de descubrimiento de información relacional. Para conseguir una herramienta de este tipo se debían cumplir varios requisitos. Por una parte, resultaba necesario disponer de una gramática que expresase las consultas a evaluar de una forma cercana a las propias estructuras sobre las que iba a trabajar. Y gracias a la capacidad expresiva de los grafos con propiedades, hemos presentado una herramienta de consulta que se puede expresar de forma natural por medio de un grafo con propiedades. Además, era necesario dotar al conjunto de consultas generadas por esta gramática de una base bien fundamentada de propiedades que nos asegurasen que, al ser usadas como predicados lógicos sobre grafos, se comportaban de manera coherente y robusta. Este resultado se ha obtenido presentando las relaciones existentes entre la estructura topológica de la consulta y las relaciones de implicación por medio del refinado. Además, era necesario, ya que en última instancia también las usaremos para generar métodos automáticos de aprendizaje, que las consultas pudiesen ser modificadas de manera controlada por medio de operadores atómicos que tradujesen el control topológico en un control lógico. En este sentido, se ha introducido una primera familia de refinamientos (un refinamiento actúa como una partición de una consulta) que permiten construir a partir de una consulta inicial (que puede ser vacía) una colección ordenada de consultas que recorren las diversas opciones de verificación, formando un retículo completo de consultas.

Debido a que cualquier estructura de datos relacional puede ser vista como un grafo y a que cualquier consulta puede ser vista como la búsqueda de un patrón, la mayoría de lenguajes de consulta en bases de datos pueden ser vistos como herramientas (quizás primitivas) de consulta de patrones en grafos con propiedades. En el capítulo 3 también se han analizado algunas de las herra-

mientas de consulta existentes, así como la viabilidad para ser utilizadas en procedimientos automáticos. Una de las herramientas analizadas, los grafos de selección, permite evaluar registros en bases de datos relacionales a través de patrones acíclicos que pueden ser refinados a partir de operaciones básicas, permitiendo obtener patrones complementarios en cada caso. Para ello, no requiere una proyección exacta del patrón que representa el grafo de selección sobre el subgrafo a evaluar, sino el cumplimiento de una serie de predicados expresados a través de dicho patrón. Debemos señalar que si se exige una proyección a la hora de realizar la verificación de un patrón se complica la tarea de evaluar la no existencia de determinados elementos. Concretamente, los grafos de selección, evalúan la existencia / no existencia de caminos incidentes al registro bajo evaluación (solo son capaces de evaluar registros individuales), para ello se verifica si se cumple una conjunción de predicados sobre caminos que parten del registro analizado, lo cual puede ser visto como la evaluación de existencia de un árbol enraizado en el nodo que representa el registro bajo evaluación.

El Property Query Graph, la herramienta presentada en este capítulo, extiende el concepto de grafo de selección permitiendo la evaluación de subgrafos generales, más allá de un único nodo, predicados abiertos a través de la definición de un lenguaje sobre los elementos del grafo y patrones cíclicos. Como se convierte en un requisito no usar una proyección para la verificación de un patrón, estos objetivos los hemos conseguido extendiendo la forma de evaluación, que puede ser vista como la evaluación de un árbol enraizado por cada nodo presente en el patrón. A pesar de que por cada nodo de un PQG se evalúa la existencia de un nodo que cumpla con las condiciones impuestas por su predicado y las aristas en las que participa, al permitir que las aristas se identifiquen con caminos en el grafo (Regular Pattern Matching) se produce la evaluación de un árbol por cada nodo, y no de una simple estrella. Es a través de las intersecciones que se producen entre los diversos árboles y de las restricciones impuestas en los nodos como se permite la evaluación de patrones cíclicos en los PQG.

Como hemos comentado, al igual que los grafos de selección, los PQG se pueden modificar y construir a partir de refinamientos, pero a diferencia del caso simple de los grafos de selección, normalmente los refinamientos no son binarios, ya que su aplicación puede modificar más de un predicado en el patrón, dando

lugar a conjuntos de tamaño 2^k (siendo k el número de predicados modificados). Tal y como muestra el capítulo 4, esto no supone ningún problema a la hora de ser utilizados en la construcción de modelos de aprendizaje, como los árboles de decisión, ya que éstos no tienen porqué ser binarios. A través de la definición de determinadas operaciones de simplificación y equivalencia, los refinamientos mostrados pueden ser simplificados dando lugar a herramientas sencillas que permiten expresar consultas complejas en grafos.

En general, los refinamientos dan lugar a particiones encajadas de las estructuras que evalúan, lo que los convierte en herramientas ideales para procedimientos de caja blanca. Tras haber llevado a cabo una primera implementación como prueba de concepto (pero totalmente funcional), se ha demostrado experimentalmente que los PQG son viables bajo condiciones suaves y que cumplen con los objetivos planteados de extensión de las herramientas existentes.

Un uso explícito de estas capacidades se muestran en el capítulo 4, con la presentación del algoritmo PQG-ID3, que hace uso de los Property Query Graphs como herramientas de test para la construcción de un árbol de decisión siguiendo los fundamentos del famoso algoritmo ID3. En los resultados de los experimentos llevados a cabo, se muestra que PQG-ID3 es capaz de extraer patrones interesantes que pueden ser utilizados en tareas de aprendizaje complejas. Para ello, basta considerar los PQG contenidos en las hojas como nuevos atributos descubiertos por el algoritmo. De esta forma, además de construir un árbol clasificador, el algoritmo es capaz de descubrir patrones que caracterizan diferentes estructuras en el grafo (*graph pattern mining*) y que pueden ser utilizados como atributos de las estructuras que clasifican en tareas posteriores (*feature extraction*).

El algoritmo MRDTL presentado en el apartado 4.2.2 puede ser visto como un caso particular del algoritmo PQG-ID3 en el que sólo se permiten PQG con forma de árbol (ya que hacen uso de grafos de selección) y donde sólo se permite aprendizaje a partir de estructuras formadas por un único nodo. En este sentido, PQG-ID3 supone un salto adelante en una línea de trabajo iniciada hace años y que se consideraba abierta desde entonces. Como curiosidad, hemos de decir que el trabajo realizado sobre PQG-ID3 fue realizado de manera completamente independiente, y solo cuando se estaba escribiendo esta memoria pudimos ponerlo en relación con los grafos de selección y el algoritmo MRDTL.

Como se vio a lo largo del capítulo, el principal problema que presentan los algoritmos de construcción de árboles de decisión multi-relacionales es que el espacio de hipótesis es extremadamente grande. Para solventar este problema se pueden proponer varias soluciones. Por un lado (y como extensión a la propuesta que se realizó con MRDTL-2), se puede analizar de manera estadística la frecuencia de aparición de ciertas estructuras atendiendo a las propiedades que intervienen y a las restricciones asociadas con el fin de reducir el número de posibles refinamientos a aplicar en cada caso y así reducir el coste de la búsqueda del mejor refinamiento. Todo este análisis previo hace uso de las diversas medidas que se han introducido en el capítulo 2 (y que extienden las medidas de frecuencia más simples que se usan en el caso de MRDTL-2). Por otro lado, se pueden crear familias de refinamientos más complejos (por ejemplo, combinar el refinamiento *añadir arista* con *añadir propiedad a una arista* en un solo paso) para de esta manera reducir el número de pasos para obtener PQG complejos y ampliar la reducción de impureza que suponen los pasos atómicos que son menos informativos. Si se lleva a cabo esta última opción de manera adecuada (unificando los refinamientos en función de la frecuencia de aparición de estructuras en el grafo) se puede conseguir que el algoritmo se acerque de manera más rápida a la solución. En ambos casos se consigue una mejora en la eficiencia sacrificando la posibilidad de cubrir un espacio de hipótesis más amplio (pero que probablemente ofrece alternativas en las que la reducción de impureza es menor). En este sentido, en este trabajo se ha ofrecido un conjunto minimal de refinamientos bien construidos, pero debe tenerse en cuenta que no se ofrecen con la intención de que sea óptimo para ciertas tareas de aprendizaje.

El segundo gran problema que tiene el agoritmo PQG-ID3 (y que es heredado por todos los algoritmos inspirados en ID3) es la imposibilidad de deshacer las decisiones tomadas durante la construcción del árbol. De tal manera que las opciones de refinamiento en un paso determinado del algoritmo dependen de los refinamientos elegidos en pasos anteriores y determinan, hasta cierto punto, las opciones futuras. Para solucionar este problema, es habitual utilizar algún procedimiento de *backtracking* que permita deshacer decisiones si han desembocado en un mal resultado, o algún procedimiento de *Beam-Search*, como el utilizado en el algoritmo GBI presentado en el apartado 4.2.3, que permita tomar varias decisiones en paralelo, y finalmente seleccionar la que haya derivado en una mejor solución.

En consecuencia, los modelos de consulta en grafos basados en PQG permiten obtener herramientas potentes y sencillas, de complejidad controlada, idóneas para su construcción automática y para ser utilizadas en tareas de caja blanca sobre información multi-relacional, debido en parte a las buenas propiedades relacionadas con complementariedad y contención de consultas. Además, la combinación del tipo de árboles de decisión PQG a través de modelo agregados, como Random Forest, puede conseguir muy buenos resultados a la hora de llevar a cabo clasificación automática (aunque con ello se reduzca la capacidad interpretativa de los modelos obtenidos).

6.2. De la inmersión semántica

El objetivo de este último capítulo ha sido el de ofrecer la posibilidad de llevar a cabo tareas de aprendizaje automático relacional a través de algoritmos más tradicionales haciendo una selección automática de atributos (*feature extraction*), y manteniendo las estructuras relacionales mediante la generación de un conjunto de entrenamiento adecuado. De esta forma, y como complemento a la aproximación presentada en los dos capítulos anteriores, buscamos analizar qué opciones ofrecen los algoritmos tradicionales cuando deseamos no perder las estructuras enriquecidas propias de la información relacional.

Si existe un elemento (un subgrafo) que está inmerso en una base de datos (un grafo con propiedades) la tarea de construir atributos para el aprendizaje a partir de las relaciones que presenta en la estructura global puede ser muy complicada. La aproximación que se presenta en este capítulo pasa por construir una representación vectorial de cada elemento en el sistema a partir de un muestreo de la información presente en la red. De esta manera, evitamos por un lado el trabajo manual de selección de los atributos a tener en cuenta y, por otro, conseguimos que el algoritmo de aprendizaje a utilizar se alimente de una representación obtenida a partir de información global disponible.

En comparación con otras tareas de aprendizaje automático, hay pocos trabajos que hayan utilizado codificadores neuronales para realizar inmersiones de grafos con propiedades, o estructuras similares, en espacios vectoriales. Nuestra metodología ha buscado usar arquitecturas simples para obtener representaciones vectoriales que mantienen las características semánticas y topológicas del

grafo original. Además, se ha demostrado experimentalmente que con las inmersiones obtenidas se pueden obtener conexiones semánticas que no aparecen explícitamente en el grafo original (debido a incompletitud en los datos almacenados, o a incoherencias en los mismos), o incluso optimizar tareas de consulta en bases de datos.

Hemos comprobado que las características geométricas de las estructuras formadas por las inmersiones de nodos y aristas en el nuevo espacio vectorial pueden ayudar a asignar tipos o propiedades faltantes a los elementos del grafo original (usando medidas relacionadas con distancia, linealidad, o agrupación, entre otras), o pueden incluso ayudar a identificar nuevas relaciones entre elementos que no están presentes explícitamente. Esta funcionalidad puede ser de gran utilidad en procesos que trabajan con grandes conjuntos de datos relacionales, donde la incompletitud de los datos es un obstáculo habitual.

Además, como se ha observado a partir de las pruebas de evaluación, el rendimiento y la precisión de las tareas de aprendizaje automático sobre estas representaciones vectoriales pueden proporcionar información sobre la estructura semántica del conjunto de datos en sí, y no sólo sobre los algoritmos en uso. Por ejemplo, la confusión de algunos nodos / aristas en tareas de clasificación puede darnos información sobre la necesidad de realizar un ajuste en el esquema de datos para reflejar las características semánticas correctamente. Un informe detallado sobre cómo los diferentes tipos, propiedades, y clusters se superponen y confunden en la inmersión sería de utilidad para tomar decisiones relacionadas con la normalización de los esquemas de datos, algo de lo que carecen casi todas las propuestas actuales de análisis.

Es evidente que el tamaño del conjunto de entrenamiento y de la ventana de selección influyen positivamente en la capacidad de aplicación de la inmersión resultante, pero estas influencias deben ser estudiadas a mayor profundidad, ya que pueden arrojar claves para la automatización de los parámetros de la inmersión.

Además, en este capítulo se ha explorado cómo las estructuras vectoriales pueden usarse para recuperar información de grafos con propiedades, como muestran los experimentos de *Entity Retrieval* y de *caminos tipados*. Es probable que buscar estructuras complejas en el espacio proyectado sea más sencillo que en el espacio original. De hecho, el uso de una segunda capa de modelos

de aprendizaje después de la codificación neuronal puede mejorar los resultados de varias tareas relacionadas con la recuperación de información en grafos semánticos. Los resultados en este trabajo muestran que esta es una línea de investigación que vale la pena ser considerada. A pesar de que no se han llevado a cabo suficientes experimentos en cuanto a consultas a larga distancia a través de los vectores representantes en el nuevo espacio, los resultados obtenidos muestran que los tiempos de consulta pueden ser reducidos considerablemente sacrificando la optimalidad. Este tipo de consultas son muy costosas en las bases de datos, y a pesar de que las bases de datos en grafo han ayudado a reducir su coste computacional siguen presentando grandes problemas de eficiencia.

Frente a otras aproximaciones en la misma dirección, este trabajo presenta la novedad de trabajar con contextos semánticos más generales, y no solo con caminos aleatorios, que suponen una linealización de la estructura del grafo original. Pero estas no son las únicas opciones para llevar a cabo codificaciones de grafos con propiedades por medio de redes neuronales. Como se planteará en el apartado 6.3, podemos conseguir codificaciones vectoriales de grafos con propiedades haciendo uso de autocodificadores neuronales, de tal manera que el codificador neuronal aprenderá la función identidad para los elementos del grafo, desligando la codificación de la función que relaciona a los elementos con su contexto.

Con este trabajo hemos dado un marco inicial para realizar tareas de aprendizaje automático a partir de grafos con propiedades en las que se tiene en cuenta información del grafo completo para codificar cada elemento. Esta nueva representación de los grafos con propiedades permite trabajar con datos relacionales almacenados en casi cualquier sistema de persistencia de manera vectorial, aprovechando la potencia que tienen actualmente los procesadores y GPUs para trabajar con este tipo de estructuras.

Debe señalarse que durante la revisión de este documento se han publicado nuevas herramientas basadas en las arquitecturas de *Word2Vec* que optimizan el proceso de aprendizaje de semánticas latentes a partir de lenguaje natural [34]. A pesar de la probable mejora que estas herramientas supondrían en nuestra metodología, hemos decidido no tenerlas en cuenta ya que no suponen un cambio en la parte fundamental de nuestros resultados, aunque sí mejoraría, posiblemente, la carga de cálculo asociada que han conllevado los experimentos

realizados.

6.3. Trabajo Futuro

En esta última sección queremos mostrar algunas de las nuevas líneas abiertas a partir de la investigación desarrollada para esta memoria. Algunas ya están siendo estudiadas, mientras que otras representan simples ideas que han ido surgiendo durante la ejecución del trabajo y han quedado apuntadas para ser abordadas cuando sea posible. En lo posible, se intentará mantener el orden natural que se ha seguido en la memoria.

La potencia conseguida a través de la definición de un lenguaje sobre los elementos de un grafo como el presentado en la sección 3.4 ha permitido construir una herramienta de descubrimiento que generaliza y potencia al estándar en construcción de árboles de decisión multi-relacionales. Sin embargo, la restricción que hemos impuesto de que dichos predicados sólo puedan evaluar nodos, aristas o caminos, implica que el patrón global represente un predicado que finalmente es una conjunción de predicados relativamente simples. Este tipo de patrones se construyen a través de alguna estructura en forma de grafo que unifique dichos predicados, varios árboles en el caso de PQG-ID3. En un PQG, los predicados que lo componen tiene una perspectiva de nodo (el PQG está constituido por un predicado por cada nodo que posee), pero se podrían construir PQGs que evaluaran a partir de otra estructura, por ejemplo, PQGs en los que se asigna un predicado por cada ciclo de longitud 3 (perspectiva de triángulos). Esta limitación sugiere que el concepto de patrón, como se ha concebido hasta ahora, no es lo suficientemente general para expresar predicados potentes y flexibles sobre datos relacionales. Una definición de patrón recursiva, que podría llevar a una redefinición/extensión del concepto de grafo a través de la recursión de estructuras por niveles, podría permitir expresar un conjunto mayor de patrones sin perder potencia, flexibilidad, ni la capacidad de poder ser construidos a través de refinamientos complementarios como los presentados.

En los PQG, el signo de los nodos y aristas tienen una interpretación intuitiva clara: los elementos positivos deben existir en el grafo bajo evaluación, y los negativos imponen restricciones de no existencia. Debido a que añadir restricciones a una condición de no existencia da como resultado una condición

menos restrictiva, no han sido susceptibles de ser refinados a través de los métodos presentados. Sin embargo, no es razón para no investigar posibles vías de refinamiento de patrones a través de los elementos negativos. Para ello, bastaría plantear este tipo de predicados en base a disyunciones, de esta forma la situación sería la inversa y las restricciones negativas serían susceptibles de ser refinadas. Así pues, para mejorar la utilidad de los PQG, y de las herramientas que se deriven de ellos, una vía de trabajo debe ser ahondar en la generación de familias de refinamientos que amplíen la capacidad expresiva de los PQG que se pueden construir automáticamente.

Los avances conseguidos en el aprendizaje de árboles de decisión multi-relacionales a través de PQG deben ser aprovechados por la familia de métodos *ensemble* y en concreto por el método Random Forest. Como hemos comentado, la capacidad interpretativa de los árboles de decisión se diluye cuando varios árboles son combinados para explicar un mismo resultado, pero pueden ampliar enormemente su capacidad predictiva. Otra opción que se deriva del uso de métodos combinados con los árboles de decisión que usan PQG es que los árboles obtenidos a través de los métodos presentados en esta memoria poseen tests en sus nodos hoja que evalúan patrones semánticos completos, por lo que una opción sería combinarlos de manera probabilística para dar lugar a patrones combinados que puedan ser interpretados como herramientas de decisión probabilística, lo que abre una línea interesante dentro del aprendizaje relacional de caja blanca. Incluso inspirarse en la existencia de arboles de decisión difusos para generar PQGs difusos.

Los PQG presentados en el capítulo 3 representan predicados sobre subgrafos con propiedades que son capaces de evaluar características más allá de las propiedades estructurales y semánticas del subgrafo en cuestión, ya que permiten expresar restricciones en el entorno de dicho subgrafo (el entorno de un subgrafo puede llegar ser todo el grafo en el que se encuentra inmerso, si se utiliza el predicado adecuado). Esta característica, sumada a las ya expuestas en esta memoria, convierten a los PQG en descriptores de estructuras relacionales (hayan sido construidos automáticamente o diseñados manualmente por expertos en el área), erigiéndose como candidatos aptos para ser utilizados como atributos adicionales en tareas de aprendizaje relacional. Además, y como ya se ha comentado, la complejidad en el método PQG-ID3 puede ser reducida

haciendo uso de análisis estadísticos para evaluar la frecuencia de aparición de diferentes patrones en el grafo y de esta forma reducir los posibles refinamientos disponibles en cada paso, o combinar varios refinamientos en uno.

Las mejoras en eficiencia en consultas a larga distancia planteadas en el apartado 5.4.6 merecen ser evaluadas a mayor profundidad y comparadas con otros métodos similares. Algunos resultados relacionados con el análisis semántico de grafos con propiedades no han sido llevados a cabo en profundidad y no se han presentado en esta memoria aunque se prevé sean presentados en trabajos posteriores. Opciones como muestrear el contexto de las aristas, realizar una inmersión de las mismas y a partir de ésta inferir una inmersión para los nodos no han sido tenidas en cuenta y pueden ofrecer resultados interesantes.

Con respecto a la inmersión de grafos con propiedades en espacios vectoriales a través de codificadores neuronales, se debe tener en cuenta que, al habernos inspirado en las arquitecturas correspondientes a *Word2Vec*, la función que dicho codificador trata de aprender relaciona cada nodo con su contexto y esto determina, por tanto, la inmersión obtenida. En este trabajo no se ha planteado que dicho codificador podría aprender otras funciones, pero consideramos que es un punto a tener en cuenta ya que la función que aprende el codificador es determinante a la hora de utilizar la inmersión posterior. Por ejemplo, si utilizáramos la función identidad (en este caso la red sería un autocodificador) podríamos obtener una inmersión *aséptica*, no determinada por ningún criterio previo. Esto permitiría evitar los problemas derivados por la definición de los contextos en éste y en otros trabajos relacionados con la inmersión de grafos a través de caminos aleatorios. Otra opción podría ser utilizar patrones como los PQG para realizar la codificación, de tal manera que, dada una estructura, la función a aprender por el codificador la relacionara con su *PQG asociado*, de esta forma la codificación obtenida reflejaría la semántica asociada al PQG utilizado para la inmersión y la representación resultante podría ser óptima si el aprendizaje supervisado (clasificación), o no supervisado (clusterización), posterior está relacionado con la estructura mostrada en el PQG. Sin duda, las posibilidades de mezclar la expresividad que aportan patrones como los PQG y la eficiencia y rendimiento que aportan las representaciones vectoriales mostradas son amplias y prometedoras.

Durante la concepción, implementación y experimentación de este trabajo

han ido abriéndose nuevas vías que pueden ser consideradas para analizar las características de las inmersiones obtenidas.

Una primera consideración a tener en cuenta está relacionada con la manera de construir el conjunto de entrenamiento que es consumido por el codificador neuronal para obtener la representación vectorial de un grafo con propiedades. En los experimentos realizados la construcción del conjunto de entrenamiento ha sido totalmente aleatoria, es decir, todos los nodos tienen la misma probabilidad de ser muestreados, al igual que todas sus propiedades y vecinos. Ésta puede no ser la manera más adecuada dependiendo del tipo de actividad que se desee realizar con la inmersión resultante. Por ejemplo, puede ser favorable construir el conjunto de entrenamiento de manera que aquellos nodos que posean una mayor riqueza semántica tengan más probabilidad de entrar en el conjunto de entrenamiento, lo que puede contribuir a que regiones inicialmente menos probables de ser consideradas compensen este hecho.

Otra línea a tener en cuenta es la de construir una red neuronal que trabaje con los contextos de un elemento como entrada (en formato one-hot) y aprenda a devolver una propiedad determinada de éste como salida de la misma, es decir, conectar el clasificador/regresor neuronal directamente con el codificador, para de esta forma aprender la codificación adecuada y la clasificación/regresión a partir de ésta de manera simultánea. De igual forma, sería interesante pensar en codificadores neuronales que hacen uso de redes neuronales recurrentes para poder analizar el comportamiento de información relacional dinámica.

También cabe destacar que queda abierta la posibilidad de trabajar con propiedades continuas en nodos y aristas, una característica no presente en los datasets utilizados, pero que debe ser considerada para ampliar la capacidad de la metodología presentada. Tanto para el caso de los PQG como para el caso de las inmersiones hay mecanismos directos para incluir la presencia de propiedades continuas, queda como trabajo comenzar probando con estos mecanismos más evidentes y medir posteriormente hasta qué punto se pueden tener en cuenta otras aproximaciones.

En resumen, la investigación presentada en esta memoria ha abierto numerosas líneas de trabajo en diversas áreas conectadas. Las más evidentes se han dado en la formalización de estructuras relacionales, formalización de procedimientos de construcción de consultas sobre las mismas, el aprendizaje automático

y el descubrimiento de información relacional, la extracción de características, aprendizaje de la representación, y análisis y normalización de datos. Algunas de ellas han sido presentadas aquí pero, sin duda y a la vista de los resultados obtenidos, surgirán nuevos retos en forma de ideas a partir este trabajo. Por ello, nos complace haber presentado una tesis en la que, a pesar de haber abordado con meticulosidad los objetivos iniciales, se han abierto más interrogantes que vías se han cerrado. Sin duda, esta profusión de posibles vías de continuidad demuestra que el estudio de sistemas de información relacional se puede convertir en una fructífera línea de investigación a la que merece la pena prestar atención.

Apéndice A

Implementaciones

El software desarrollado para llevar a cabo los experimentos presentados en esta memoria está disponible como software libre en una colección de repositorios digitales de acceso abierto. Concretamente, se han desarrollado dos librerías de software para llevar a cabo tareas relacionadas con el aprendizaje automático a partir de grafos con propiedades y con la inmersión de estas mismas estructuras en espacios vectoriales. En ambos casos, las librerías han sido programadas en Python y están diseñadas para consumir los datos de entrada a partir de una base de datos Neo4j.

Pasamos ahora a describir brevemente ambas librerías y a detallar el repositorio en el que se encuentran disponibles.

- *Property Query Graphs*, corresponde a la implementación de los modelos desarrollados en los capítulos 3 y 4, relacionados con consulta de patrones en grafos con propiedades y construcción de árboles de decisión multi-relacionales. Esta librería implementa los Property Query Graphs a través de una herramienta de evaluación, y el algoritmo PQG-ID3, permitiendo llevar a cabo aprendizaje de árboles de decisión multi-relacionales a partir de datos estructurados en forma de grafo (concretamente, a partir de datos almacenados en una base de datos Neo4j). Se encuentra disponible en `https://github.com/palmagro/pqg`.

- *Graph2Vec*, corresponde a la implementación de los modelos desarrollados en el Capítulo 5, relacionados con inmersiones vectoriales de los elementos de un grafo con propiedades y con el aprendizaje automático a partir

las mismas. Se encuentra disponible en `https://github.com/palmagro/node2vec`.

En las direcciones indicadas se pueden encontrar ejemplos de aplicación de ambas librerías sobre redes semánticas correspondientes a conjuntos de datos reales que por motivos de limitaciones del formato de presentación habitual no han sido incluidos en esta memoria.

Bibliografía

[1] Cypher into patterns. http://neo4j.com/docs/stable/cypher-intro-patterns.html.

[2] Cypher introduction. http://neo4j.com/docs/stable/cypher-introduction.html.

[3] D2r server: Accessing databases with sparql and as linked data. http://d2rq.org/d2r-server.

[4] The sparql2xquery framework. http://www.dblab.ntua.gr/~bikakis/SPARQL2XQuery.html.

[5] Sparqlimplementations. https://www.w3.org/wiki/SparqlImplementations.

[6] Xml 1.0 origin and goals. https://www.w3.org/TR/REC-xml/#sec-origin-goals.

[7] Xml: The angle bracket tax. https://blog.codinghorror.com/xml-the-angle-bracket-tax/.

[8] Xpath - retrieving nodes from an xml document. http://sqlmag.com/xml/xpath151retrieving-nodes-xml-document.

[9] *Fast Graph Pattern Matching*, April 2008.

[10] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.

[11] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. Connectionist models and their implications: Readings from cognitive science. chapter A Learning Algorithm for Boltzmann Machines, pages 285–307. Ablex Publishing Corp., Norwood, NJ, USA, 1988.

[12] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Satya Sanket Sahoo, Amit Sheth, and I. Budak Arpinar. *Template Based Semantic Similarity for Security Applications*, pages 621–622. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[13] Faisal Alkhateeb, Jean-Francois Baget, and Jérôme Euzenat. *RDF with regular expressions*. PhD thesis, INRIA, 2007.

[14] Noga Alon and Raphael Yuster. On a hypergraph matching problem. *Graphs and Combinatorics*, 21(4):377–384, 2005.

[15] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

[16] Kartik Anand and Ginestra Bianconi. Entropy measures for networks: Toward an information theory of complex topologies. *Physical Review E*, 80(4):045102, 2009.

[17] Renzo Angles. A comparison of current graph database models. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*, ICDEW '12, pages 171–177, Washington, DC, USA, 2012. IEEE Computer Society.

[18] Renzo Angles, Pablo Barceló, and Gonzalo Ríos. A practical query language for graph dbs. In *7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, 2103.

[19] Renzo Angles, Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. Benchmarking database systems for social network applications. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 15:1–15:7, New York, NY, USA, 2013. ACM.

[20] T.M. Apostol. *Mathematical Analysis*. 1957.

[21] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection, 2009.

[22] Anna Atramentov, Hector Leiva, and Vasant Honavar. *A Multi-relational Decision Tree Learning Algorithm – Implementation and Experiments*, pages 38–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[23] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[24] Richard Barber and Austin Gibbons. Automatic annotation in multirelational information networks. 2011.

[25] Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 199–210, New York, NY, USA, 2011. ACM.

[26] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[27] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[28] George M. Bergman. *An Invitation to General Algebra and Universal Constructions*. Henry Helson, 15 the Crescent, Berkeley CA, 94708, 1998.

[29] Ginestra Bianconi, Anthony C. C. Coolen, and Conrad J. Perez Vicente. Entropies of complex networks with hierarchically constrained topologies. *Physical Review E*, 78(1):016114+, July 2008.

[30] Bahareh Bina, Oliver Schulte, Branden Crawford, Zhensong Qian, and Yi Xiong. Simple decision forests for multi-relational classification. *Decision Support Systems*, 54(3):1269–1279, 2013.

[31] Hannah Blau, Neil Immerman, and David D. Jensen. A visual language for relational knowledge discovery. Technical Report UM-CS-2002-37, Department of Computer Science, University of Massachusetts, Amherst, MA, 2002.

[32] Hendrik Blockeel and Luc De Raedt. Top-down induction of logical decision trees.

[33] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1):285 – 297, 1998.

[34] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[35] D. Bonchev. *Information Theoretic Indices for Characterization of Chemical Structures*. Chemometrics research studies series. Research Studies Press, 1983.

[36] U. S. R. Bondy, J. A.; Murty. *Graph Theory. Graduate Texts in Mathematics*. Springer, 2008.

[37] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc., 2013.

[38] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[39] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[40] I. Bratko and M. Bohanec. Trading accuracy for simplicity in decision trees, 1994.

[41] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[42] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[43] Eric Browne. The myth of Self-Describing XML, 2003.

[44] A.E. Bryson. *Applied Optimal Control: Optimization, Estimation and Control*. Halsted Press book'. Taylor & Francis, 1975.

[45] Horst Bunke, Peter Dickinson, Miro Kraetzl, Michel Neuhaus, and Marc Stettler. *Matching of Hypergraphs — Algorithms, Applications, and Experiments*, pages 131–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[46] Yang Cao, Wenfei Fan, Jinpeng Huai, and Ruizhe Huang. Making pattern queries bounded in big graphs. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 161–172, 2015.

[47] Steve Cassidy. Generalizing XPath for directed graphs. In *Proceedings for the Extreme Markup Languages conference*, 2003.

[48] Kai-Wei Chang, Scott Wen-tau Yih, Bishan Yang, and Chris Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. ACL – Association for Computational Linguistics, October 2014.

[49] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 119–128, New York, NY, USA, 2015. ACM.

[50] Jie Cheng, Christos Hatzis, Hisashi Hayashi, Mark-A. Krogel, Shinichi Morishita, David Page, and Jun Sese. Kdd cup 2001 report. *SIGKDD Explor. Newsl.*, 3(2):47–64, January 2002.

[51] Yu Cheng and Daniel Suthers. Social network analysis—centrality measures. 2011.

[52] W. J. Christmas and C Fl W. J. Christmas. Structural matching in computer vision using probabilistic reasoning, 1995.

[53] E. F. Codd. Relational database: A practical foundation for productivity. *Commun. ACM*, 25(2):109–117, February 1982.

[54] Thayne Coffman, Seth Greenblatt, and Sherry Marcus. Graph-based technologies for intelligence analysis. *Commun. ACM*, 47(3):45–47, March 2004.

[55] A. Colmerauer and P. Roussel. *La naissance de Prolog.* Editions universitaires europeennes EUE, 2014.

[56] Mariano P. Consens and Alberto O. Mendelzon. Graphlog: A visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '90, pages 404–416, New York, NY, USA, 1990. ACM.

[57] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[58] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Int. Res.*, 1(1):231–255, February 1994.

[59] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[60] Quinlan Quinlan Cs and J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[61] Balázs Csanád Csáji. Approximation with artificial neural networks. *MSc Thesis, Eötvös Loránd University (ELTE), Budapest, Hungary*, 2001.

[62] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. Graphframes: an integrated API for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, Redwood Shores, CA, USA, June 24 - 24, 2016*, page 2, 2016.

[63] Manlio De Domenico, Albert Solé-Ribalta, Emanuele Cozzo, Mikko Kivelä, Yamir Moreno, Mason A. Porter, Sergio Gómez, and Alex Arenas. Mathematical formulation of multilayer networks. *Phys. Rev. X*, 3:041022, Dec 2013.

[64] H. Decker, L. Lhotská, S. Link, M. Spies, and R.R. Wagner. *Database and Expert Systems Applications: 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings.* Number parte 1 in Lecture Notes in Computer Science. Springer International Publishing, 2014.

[65] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[66] Matthias Dehmer. Information processing in complex networks: Graph entropy and information functionals. *Applied Mathematics and Computation*, 201(1–2):82 – 94, 2008.

[67] Matthias Dehmer. A novel method for measuring the structural information content of networks. *Cybern. Syst.*, 39(8):825–842, November 2008.

[68] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA, 2014. ACM.

[69] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[70] Sašo Džeroski. Multi-relational data mining: An introduction. *SIGKDD Explor. Newsl.*, 5(1):1–16, July 2003.

[71] Frank Emmert-Streib and Matthias Dehmer. Information theoretic measures of uhg graphs with low computational complexity. *Applied Mathematics and Computation*, 190(2):1783–1794, 7 2007.

[72] Paul Erdős and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959 1959.

[73] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(5):476–491, May 1997.

[74] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 8–21, New York, NY, USA, 2012. ACM.

[75] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 8–21, New York, NY, USA, 2012. ACM.

[76] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. Adding regular expressions to graph reachability and pattern queries. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE*, pages 39–50. IEEE Computer Society, 2011.

[77] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: From intractable to polynomial time. *Proc. VLDB Endow.*, 3(1-2):264–275, September 2010.

[78] Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang, and Yinghui Wu. Graph homomorphism revisited for graph matching. *Proc. VLDB Endow.*, 3(1-2):1161–1172, September 2010.

[79] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Language, speech, and communication. MIT Press, 1998.

[80] Philip Fennell. Extremes of xml. In *XML London 2013*, 2013.

[81] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194 – 211, 1979.

[82] Scott Fortin. The graph isomorphism problem. Technical report, 1996.

[83] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121, 1980.

[84] Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

[85] Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, March 1977.

[86] Brian Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[87] Warodom Geamsakul, Takashi Matsuda, Tetsuya Yoshida, Hiroshi Motoda, and Takashi Washio. *Classifier Construction by Graph-Based Induction for Graph-Structured Data*, pages 52–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[88] Warodom Geamsakul, Tetsuya Yoshida, Kouzou Ohara, Hiroshi Motoda, Hideto Yokoi, and Katsuhiko Takabayashi. Constructing a decision tree for graph-structured data and its applications. *Fundam. Inf.*, 66(1-2):131–160, November 2004.

[89] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp. An iterative growing and pruning algorithm for classification tree design. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(2):163–174, 1991.

[90] Xavier Glorot, Antoine Bordes, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *CoRR*, abs/1301.3485, 2013.

[91] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, Inc., New York, NY, USA, 1990.

[92] Charles F. Goldfarb. The roots of sgml - a personal recollection. `http://www.sgmlsource.com/history/roots.htm`, 1996.

[93] Charles F Goldfarb. The roots of sgml: A personal recollection. *Technical communication*, 46(1):75–83, 1999.

[94] Joris Graaumans. *Usability of XML Query Languages*. PhD thesis, Proefschrift Universiteit Utrecht.

[95] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. cite arxiv:1607.00653Comment: In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[96] Andrey Gubichev and Manuel Then. Graph pattern matching: Do we have to reinvent the wheel? In *Proceedings of Workshop on GRAph Data Management Experiences and Systems*, GRADES'14, pages 8:1–8:7, New York, NY, USA, 2014. ACM.

[97] S. Gupta. *Neo4j Essentials*. Community experience distilled. Packt Publishing, 2015.

[98] J. F. Guo H. Zheng and J. Y Wang. A relational data classification algorithm with user guide.

[99] F. Harary. *Graph Theory*. Addison-Wesley, 1994.

[100] Olaf Hartig. Reconciliation of rdf* and property graphs. *CoRR*, abs/1409.3288, 2014.

[101] Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York, 2009. Autres impressions : 2011 (corr.), 2013 (7e corr.).

[102] Taher H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 517–526, New York, NY, USA, 2002. ACM.

[103] Huahai He and Ambuj K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 405–418, New York, NY, USA, 2008. ACM.

[104] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society.

[105] Yi He, Jian-chao Han, and Shao-hua Zeng. *Classification Algorithm based on Improved ID3 in Bank Loan Application*, pages 1124–1130. Springer London, London, 2012.

[106] Ivan Herman and M Scott Marshall. Graphxml—an xml-based graph description format. In *International Symposium on Graph Drawing*, pages 52–62. Springer, 2000.

[107] I. N. Herstein. *Topics in Algebra*. Ginn and Company, 1964.

[108] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[109] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[110] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[111] Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative Group Studies*, 2(2):107–124, 1971.

[112] Florian Holzschuher and René Peinl. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 195–204, New York, NY, USA, 2013. ACM.

[113] Jiewen Huang, Kartik Venkatraman, and Daniel J. Abadi. Query optimization of distributed pattern matching. In *ICDE*, 2014.

[114] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. Datalog and emerging applications: An interactive tutorial. In *Proceedings of the*

2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, pages 1213–1216, New York, NY, USA, 2011. ACM.

[115] Yann Jacob, Ludovic Denoyer, and Patrick Gallinari. Learning latent representations of nodes for classifying in heterogeneous social networks. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 373–382, New York, NY, USA, 2014. ACM.

[116] Li Ji, Wang Bing-Hong, Wang Wen-Xu, and Zhou Tao. Network entropy based on topology configuration and its computation to random networks. *Chinese Physics Letters*, 25(11):4177, 2008.

[117] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[118] Ian L Kaplan, Ghaleb M Abdulla, S Terry Brugger, and Scott R Kohn. Implementing graph pattern queries on a relational database. *Lammerce Livermore National Laboratory, Tech. Rep*, 2008.

[119] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[120] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[121] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(2):119–127, 1980.

[122] Yusuf Kavurucu, Pinar Senkul, and Ismail Hakki Toroslu. Confidence-based concept discovery in multi-relational data mining.

[123] Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 381–388. AAAI Press, 2006.

[124] Nikhil S. Ketkar, Lawrence B. Holder, and Diane J. Cook. Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, pages 71–76, New York, NY, USA, 2005. ACM.

[125] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[126] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[127] Mikko Kivelä, Alexandre Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *CoRR*, abs/1309.7233, 2013.

[128] Arno J. Knobbe. Multi-relational data mining. In *Proceedings of the 2005 Conference on Multi-Relational Data Mining*, pages 1–118, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[129] Arno J. Knobbe, Arno Siebes, Danil Van Der Wallen, and Syllogic B. V. Multi-relational decision tree induction. In *In Proceedings of PKDD' 99, Prague, Czech Republic, Septembre*, pages 378–383. Springer, 1999.

[130] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[131] Ron Kohavi and Ross Quinlan. Decision tree discovery. In *IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY*, pages 267–276. University Press, 1999.

[132] Christian Krause, Daniel Johannsen, Radwan Deeb, Kai-Uwe Sattler, David Knacker, and Anton Niadzelka. *An SQL-Based Query Language and Engine for Graph Pattern Matching*, pages 153–169. Springer International Publishing, Cham, 2016.

[133] Mark-A. Krogel and Stefan Wrobel. *Transformation-Based Learning Using Multirelational Aggregation*, pages 142–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[134] S. Kullback. *Information Theory And Statistics*. Dover Pubns, 1997.

[135] P. Laird. Weighing hypotheses: Incremental learning from noisy data. In *Proc. of the 1993 AAAI Spring Symposium on Training Issues in Incremental Learning*, pages 88–95, Stanford, California, 1993.

[136] Rolf Landauer. Computation: A fundamental physical view. *Physica Scripta*, 35(1):88, 1987.

[137] Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 529–539, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[138] Daniel T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. Wiley-Interscience, 2004.

[139] Héctor Ariel Leiva, Shashi Gadia, and Drena Dobbs. Mrdtl: A multi-relational decision tree learning algorithm. In *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003*, pages 38–56. Springer-Verlag, 2002.

[140] Juan Li. Improved multi-relational decision tree classification algorithm.

[141] Gilles Louppe. *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liege, Belgium, 10 2014. arXiv:1407.7502.

[142] L. H. Luan and G. L. Ji. Research on the decision tree classification technology, 2004.

[143] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Capturing topology in graph pattern matching. *Proc. VLDB Endow.*, 5(4):310–321, December 2011.

[144] J Kent Martin and Daniel S Hirschberg. *The time complexity of decision tree induction*. 1995.

[145] Takashi Matsuda, Hiroshi Motoda, Tetsuya Yoshida, and Takashi Washio. *Knowledge Discovery from Structured Data by Beam-Wise Graph-Based Induction*, pages 255–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[146] Warren S. McCulloch and Walter Pitts. Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.

[147] William J. McGill. Applications of information theory in experimental psychology*. *Transactions of the New York Academy of Sciences*, 19(4 Series II):343–351, 1957.

[148] Caroline R. McNulty, George F.; Shallon. Inherently nonfinitely based finite algebras. *Universal algebra and lattice theory (Puebla, 1982), Lecture Notes in Math., 1004, Berlin, New York: Springer-Verlag, pp. 206–231, doi:10.1007/BFb0063439, MR 716184*, 1983.

[149] Jim Melton and Alan R Simon. *SQL: 1999: understanding relational language components*. Morgan Kaufmann, 2001.

[150] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[151] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[152] Tomas Mikolov, Jiri Kopecky, Lukas Burget, Ondrej Glembek, and Jan Cernocky. Neural network based language models for highly inflective languages. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '09, pages 4725–4728, Washington, DC, USA, 2009. IEEE Computer Society.

[153] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[154] Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.

[155] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[156] R. Milo and et al. Network motifs: simple building blocks of complex networks, 2002.

[157] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[158] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[159] Abbe Mowshowitz and Matthias Dehmer. Entropy and the complexity of graphs revisited. *Entropy*, 14(3):559–570, 2012.

[160] 2Mr. Kushik K Rana Mr. Brijain R Patel. A survey on decision tree algorithm for classification, 2014.

[161] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Min. Knowl. Discov.*, 2(4):345–389, December 1998.

[162] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Mach. Learn.*, 52(3):239–281, September 2003.

[163] Lorenzo De Nardo, Francesco Ranzato, and Francesco Tapparo. The subgraph similarity problem. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):748–749, 2009.

[164] Phu Chien Nguyen, Kouzou Ohara, Akira Mogi, Hiroshi Motoda, and Takashi Washio. *Constructing Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction*, pages 390–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[165] Phu Chien Nguyen, Kouzou Ohara, Hiroshi Motoda, and Takashi Washio. *Cl-GBI: A Novel Approach for Extracting Typical Patterns from Graph-Structured Data*, pages 639–649. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[166] Maximilian Nickel and Volker Tresp. A three-way model for collective learning on multi-relational data.

[167] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: Scalable machine learning for linked data. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 271–280, New York, NY, USA, 2012. ACM.

[168] Vincenzo Nicosia, Regino Criado, Miguel Romance, Giovanni Russo, and Vito Latora. Controlling centrality in complex networks. *arXiv preprint arXiv:1109.4521*, 2011.

[169] Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 631–636, New York, NY, USA, 2003. ACM.

[170] Jae Dong Noh and Heiko Rieger. Random walks on complex networks. *Physical review letters*, 92(11):118701, 2004.

[171] A.B.J. Novikoff. On convergence proofs on perceptrons. 12:615–622, 1962.

[172] SH. OATES-WILLIAMS. Graphs und universal algebras. *Lecture Notes Math. 884 (1981), 351-354.*, 1981.

[173] Cristina Olaru and Louis Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets Syst.*, 138(2):221–254, September 2003.

[174] Tore Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2):159–167, 2013.

[175] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations of concepts using linear relational embedding. *IEEE Trans. on Knowl. and Data Eng.*, 13(2):232–244, March 2001.

[176] Neelamadhab Padhy and Rasmita Panigrahi. Multi relational data mining approaches: A data mining technique. *CoRR*, abs/1211.3871, 2012.

[177] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998.

[178] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[179] Gordon Plotkin. Automatic methods of inductive inference. 1972.

[180] Nataliia Pobiedina, Stefan Rümmele, Sebastian Skritek, and Hannes Werthner. *Benchmarking Database Systems for Graph Pattern Matching*, pages 226–241. Springer International Publishing, Cham, 2014.

[181] Reinhard Pöschel. Graph algebras and graph varieties. *algebra universalis*, 27(4):559–577, 1990.

[182] E. Prisner. *Graph Dynamics*. Chapman & Hall/CRC Research Notes in Mathematics Series. Taylor & Francis, 1995.

[183] J. Punin and Mukkai Krishnamoorthy. XGMML (eXtensible Graph Markup and Modeling Language) 1.0 Draft Specification., 2001.

[184] J. R. Quilan. Machine intelligence 11. chapter Decision Trees and Multi-valued Attributes, pages 305–318. Oxford University Press, Inc., New York, NY, USA, 1988.

[185] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[186] J. R. Quinlan. Simplifying decision trees. *Int. J. Man-Mach. Stud.*, 27(3):221–234, September 1987.

[187] J. R. Quinlan. Learning logical definitions from relations. *MACHINE LEARNING*, 5:239–266, 1990.

[188] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[189] Luc De Raedt and Sašo Džeroski. First-order jk-clausal theories are pac-learnable. *Artificial Intelligence*, 70(1):375 – 392, 1994.

[190] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *J. Graph Theory*, 1(4):339–363, 1977.

[191] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[192] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[193] Juan L. Reutter. *Graph Patterns: Structure, Query Answering and Applications in Schema Mappings and Formal Language Theory*. PhD thesis, The school where the thesis was written, Laboratory for Foundations of Computer Science School of Informatics University of Edinburgh, 2013.

[194] Pedro Ribeiro and Fernando Silva. G-tries: An efficient data structure for discovering network motifs. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1559–1566, New York, NY, USA, 2010. ACM.

[195] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.

[196] Carlos R. Rivero and Hasan M. Jamil. Anatomy of graph matching based on an xquery and RDF implementation. *CoRR*, abs/1311.2342, 2013.

[197] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[198] Marko A. Rodriguez. A multi-relational network to support the scholarly communication process. *CoRR*, abs/cs/0601121, 2006.

[199] Marko A. Rodriguez. The gremlin graph traversal machine and language. *CoRR*, abs/1508.03843, 2015.

[200] Marko A. Rodriguez and Peter Neubauer. The graph traversal pattern. *CoRR*, abs/1004.1001, 2010.

[201] Marko A. Rodriguez and Peter Neubauer. A path algebra for multi-relational graphs. *CoRR*, abs/1011.0390, 2010.

[202] Marko A. Rodriguez and Joshua Shinavier. Exposing multi-relational networks to single-relational network analysis algorithms. *CoRR*, abs/0806.2274, 2008.

[203] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - a survey. *Trans. Sys. Man Cyber Part C*, 35(4):476–487, November 2005.

[204] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[205] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[206] Mohamed Rouane-Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. Relational concept analysis: mining concept lattices from multi-relational data. *Annals of Mathematics and Artificial Intelligence*, 67(1):81–108, 2013.

[207] S. Ruggieri. Efficient c4.5. *IEEE Trans. on Knowl. and Data Eng.*, 14(2):438–444, March 2002.

[208] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[209] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[210] S. Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology, 1991.

[211] Marcos Salganicoff, Lyle H. Ungar, and Ruzena Bajcsy. Active learning for vision-based robot grasping. *Machine Learning*, 23(2):251–278, 1996.

[212] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[213] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[214] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017.

[215] Toby Segaran, Colin Evans, Jamie Taylor, Segaran Toby, Evans Colin, and Taylor Jamie. *Programming the Semantic Web*. O'Reilly Media, Inc., 1st edition, 2009.

[216] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[217] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001.

[218] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[219] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 39–52, New York, NY, USA, 2002. ACM.

[220] Jitesh Shetty and Jafar Adibi. Discovering important nodes through graph entropy the case of enron email database. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD '05, pages 74–81, New York, NY, USA, 2005. ACM.

[221] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1986.

[222] Ajit P. Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 650–658, New York, NY, USA, 2008. ACM.

[223] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 926–934. Curran Associates, Inc., 2013.

[224] Ilya Sutskever, Joshua B. Tenenbaum, and Ruslan R Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1821–1828. Curran Associates, Inc., 2009.

[225] Anand Takale. *Constructing Predictive Models to Assess the Importance of Variables in Epidemiological Data Using A Genetic Algorithm System employing Decision Trees*. PhD thesis, UNIVERSITY OF MINNESOTA, 2004.

[226] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of*

the 24th International Conference on World Wide Web, WWW '15, pages 1067–1077, New York, NY, USA, 2015. ACM.

[227] Yuanyuan Tian and Jignesh M. Patel. Tale: A tool for approximate large graph matching. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 963–972, Washington, DC, USA, 2008. IEEE Computer Society.

[228] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 737–746, New York, NY, USA, 2007. ACM.

[229] Godfried T. Toussaint. Bibliography on estimation of misclassification. *IEEE Trans. Information Theory*, 20(4):472–479, 1974.

[230] Vaibhav Tripathy. A comparative study of multi-relational decision tree learning algorithm.

[231] Robert P. Trueblood and John N. Lovett, Jr. *Data Mining and Statistical Analysis Using SQL*. Apress, Berkely, CA, USA, 2001.

[232] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976.

[233] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695–703, September 1988.

[234] Paul E. Utgoff. Incremental induction of decision trees. *Mach. Learn.*, 4(2):161–186, November 1989.

[235] Anneleen Van Assche. A Random Forest Approach to Relational Learning.

[236] Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. Pgql: a property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, page 7. ACM, 2016.

[237] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.

[238] Denny Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 1063–1064, New York, NY, USA, 2012. ACM.

[239] Radim Řehůřek. *Scalability of Semantic Analysis in Natural Language Processing*. PhD thesis, Masaryk University, May 2011.

[240] Hongzhi Wang and Jianzhong Li. Gxquery: Extending xquery for querying graph-structured XML data. *CIT*, 19(2):83–91, 2011.

[241] Huazheng Wang, Bin Gao, Jiang Bian, Fei Tian, and Tie-Yan Liu. Solving verbal comprehension questions in IQ test by knowledge-powered word embedding. *CoRR*, abs/1505.07909, 2015.

[242] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. 2015.

[243] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, July 2003.

[244] Scott White and Padhraic Smyth. Algorithms for estimating relative importance in networks. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 266–275, New York, NY, USA, 2003. ACM.

[245] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

[246] David Wood, Markus Lanthaler, and Richard Cyganiak. RDF 1.1 concepts and abstract syntax, February 2014.

[247] Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, April 2012.

[248] Huayu Wu, Tok Wang Ling, Gillian Dobbie, Zhifeng Bao, and Liang Xu. Reducing graph matching to tree matching for XML queries with ID references. In *Database and Expert Systems Applications, 21th International Conference,DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part II*, pages 391–406, 2010.

[249] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2014.

[250] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Learning multi-relational semantics using neural-embedding models. *CoRR*, abs/1411.4072, 2014.

[251] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *Proceedings of the 2004 European Conference on Constraint-Based Mining and Inductive Databases*, pages 172–195, Berlin, Heidelberg, 2005. Springer-Verlag.

[252] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. *CrossMine: Efficient Classification Across Multiple Database Relations*, pages 172–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[253] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Efficient classification across multiple database relations: A crossmine approach. *IEEE Trans. on Knowl. and Data Eng.*, 18(6):770–783, June 2006.

[254] Kenichi Yoshida, Hiroshi Motoda, and Nitin Indurkhya. Graph-based induction as a unified learning framework. *Applied Intelligence*, 4(3):297–316, 1994.

[255] R. Zass and A. Shashua. Probabilistic graph and hypergraph matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, June 2008.

[256] Wei Zhang. Multi-relational data mining based on higher-order inductive logic programming. *2013 Fourth Global Congress on Intelligent Systems*, 2:453–458, 2009.

[257] H. Zheng. Research on the relational data classification algorithm based on background knowledges.

[258] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[259] Lei Zou, Lei Chen, and M. Tamer Özsu. Distance-join: Pattern match query in a large graph database. *Proc. VLDB Endow.*, 2(1):886–897, August 2009.