Universidad Zaragoza
1542

Trabajo de Fin de Grado

KNOWGLY: Indexación inteligente de entidades en grafos de conocimiento para habilitar búsquedas semánticas

KNOWGLY: Intelligent entity indexing in Knowledge Graphs to enable Semantic Search

Autor

Samuel Daniel García Vázquez

Director

Carlos Bobed Lisbona

Grado de Ingeniería Informática

Escuela de Ingeniería y Arquitectura

2023

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Samuel Daniel García Vázquez ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de Estudios de la titulación de Grado en Ingeniería Informática (Título del Trabajo)

KNOWGLY: Indexación inteligente de entidades en grafos de conocimiento para habilitar búsquedas semánticas

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de Mayo de 2023

Fdo: Samuel

Resumen

A medida que la cantidad de información a la que todos estamos expuestos crece, se hace visible una mayor necesidad de analizarla y explotarla de una forma más eficaz. Actualmente, todos los grandes motores de búsqueda, gran parte de las redes sociales o asistentes virtuales, entre otros, requieren responder a consultas de conocimiento general o específico sobre conjuntos de datos cada vez mayores. Por ello, cualquier mejora en la precisión, y por tanto en la satisfacción de los usuarios, es primordial.

Tradicionalmente, la recuperación de información se ha basado en técnicas puramente sintácticas, como los emparejamientos de palabras clave de una consulta frente a una colección de documentos. A medida que los conjuntos de datos se disparan en tamaño y la complejidad de las consultas crece, estos métodos muestran mayores problemas de precisión y exhaustividad. La necesidad de resolver estos problemas ha despertado un gran interés por el campo de la búsqueda semántica, que consiste en tener en cuenta el contexto y el propósito de las necesidades de información de los usuarios. Este campo se encuentra estrechamente relacionado con los grafos de conocimiento, repositorios de información estructurados que codifican relaciones entre conceptos con una semántica subyacente. Esto es debido a que, además de otorgar significado semántico a la información, permiten expresar consultas en un lenguaje formal, normalmente *SPARQL*. Estos lenguajes permiten obtener resultados mediante emparejamientos exactos de patrones o condiciones específicas, permitiendo obtener así una mayor precisión.

El uso de los grafos de conocimiento por parte de usuarios, sin embargo, conlleva un gran inconveniente, debido a que la generación de consultas en lenguajes formales se aleja del paradigma tradicional de las búsquedas de texto libre, al que la gran mayoría de ellos están acostumbrados. Para sortear este problema, existen dos aproximaciones alternativas: la primera y más popular de ellas consiste en la generación programática de consultas formales a partir de los términos de dichas búsquedas, con una pérdida inherente de precisión. La segunda se basa en la conversión del grafo a un índice de documentos de un sistema de recuperación de información tradicional, sobre el cual los usuarios pueden ejecutar búsquedas de texto libre.

Esta última aproximación, que ha sido estudiada ampliamente, supone la conversión de los conceptos o entidades presentes en el grafo a documentos tradicionales, incluyendo en ellos el contenido textual de sus atributos y relaciones con otras entidades. Como inconveniente, organizar el índice para conseguir asignar una mayor importancia a cierto contenido textual frente al resto requiere un análisis de la semántica del grafo, algo inabordable de forma manual en grafos de gran tamaño.

En este trabajo hemos desarrollado *Knowgly*, un sistema completo que implementa esta segunda aproximación, desde la etapa de indexación a la de búsqueda, buscando a su vez contrarrestar sus inconvenientes. Esto lo conseguimos mediante la aplicación de métricas sobre las tripletas de información que componen el grafo a bajo nivel, teniendo gracias a ellas en cuenta su estructura semántica durante la indexación de las entidades. De esta forma, evitamos realizar análisis manuales sobre el grafo.

Agradecimientos

A mi director, Carlos Bobed, y a todos los miembros del grupo de investigación SID, por todo el apoyo que me han ofrecido durante mi estancia en su grupo y la realización de este trabajo.

A mis compañeros de clase, por haber compartido conmigo estos cuatro años.

A mis amigos, por haberme aguantado durante bastantes más años, y los que vendrán.

A mi madre y a mi hermana.

Índice

1. Introducción

La búsqueda semántica consiste en analizar el contexto y propósito de las necesidades de información tradicionales, como las que se realizan en motores de búsqueda web, permitiendo, por ejemplo, desambiguar semánticamente consultas sobre conceptos que léxicamente son cercanos o idénticos. De esta forma, se puede conseguir realizar búsquedas con una mayor precisión, o incluso responder a preguntas expresadas en lenguaje natural.

Actualmente, todos los grandes motores de búsqueda comerciales, así como redes sociales o asistentes personales en teléfonos móviles, aplican técnicas de búsqueda semántica para mejorar sus resultados u ofrecer nuevas funcionalidades, como las *infoboxes*, como se puede observar en la Figura 1.

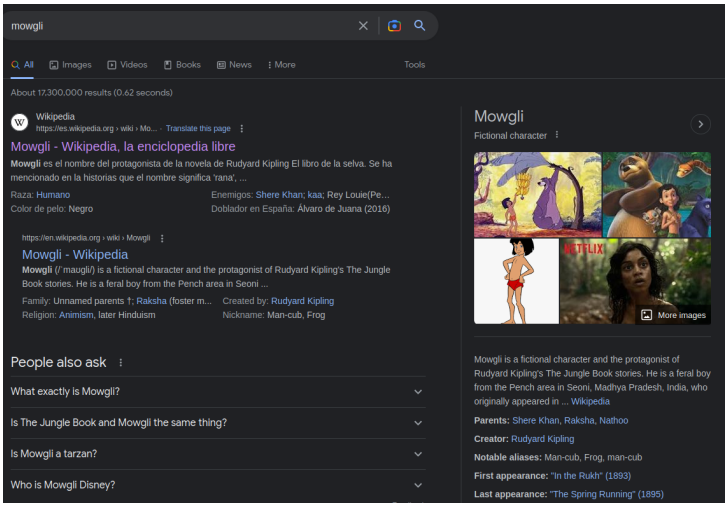

Figura 1: Ejemplo de una *Infobox* (derecha) en el motor de búsqueda *Google*, que extrae información de entidades grafos de conocimiento libres, como la *DBpedia*, y propietarios, como el grafo del conocimiento de *Google*, lanzado en 2012. Hasta ahora, *Google* aún no ha proporcionado ninguna descripción concisa sobre su funcionamiento.

1.1. Grafos de conocimiento

La tecnología fundamental de este campo son los grafos de conocimiento o *Knowledge Graphs*[1], un tipo especial de base de conocimiento (*Knowledge Base*[2]) que añade la noción de semántica. La definición más comúnmente adoptada las especifica como colecciones de tripletas $< s, p, o >$, donde s son sujetos, p son predicados y o son objetos. Como únicas restricciones, los sujetos y predicados deben ser *URIs* válidas, mientras que los objetos pueden o bien ser otras *URIs* o valores literales de un cierto tipo de dato, también denominados atributos. De manera similar a las *URLs* utilizadas en sistemas de ficheros o páginas web para identificar una localización, una *URI* en un grafo actúa como un identificador único para un recurso

[1] https://en.wikipedia.org/wiki/Knowledge_graph
[2] https://en.wikipedia.org/wiki/Knowledge_base

específico. Un ejemplo de esto se puede encontrar en la Figura 2.

Los grafos de conocimiento se fundamentan en utilizar a los sujetos como identificadores de una entidad, que pueden ser representaciones de cualquier conceptualización, como personas, objetos o hechos. De esta forma, se pueden representar relaciones arbitrarias entre distintas entidades, así como asociar propiedades a ellas mediante objetos literales.

Estos grafos, a su vez, suelen venir acompañados de una ontología, definida por Gruber [4] como una especificación formal y explicita de una conceptualización compartida. Estas ontologías abren la posibilidad de definir semánticamente de la forma tan exhaustiva como se desee, normalmente a través de restricciones lógicas, un dominio específico. Este enorme potencial ofrecido por las ontologías es exactamente a lo que los grafos de conocimiento deben su nombre, ya que permiten modelar cualquier conjunto de conocimientos sobre la realidad.

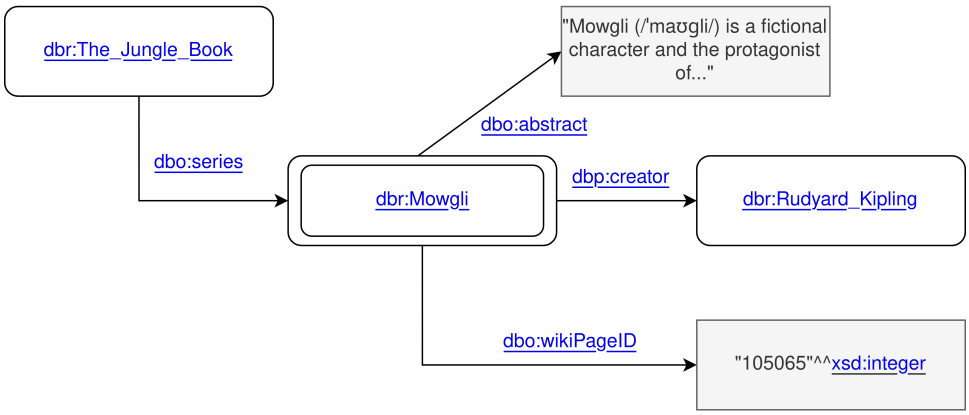

Figura 2: Ejemplo de la entidad *Mowgli*, personaje de *El libro de la selva*. Los sujetos y objetos aparecen en los recuadros, donde los sujetos son *URIs* y los objetos son *URIs* o valores literales, y las aristas dirigidas etiquetadas con *URIs* son los predicados. En este ejemplo, se encuentra relacionada con la entidad correspondiente a su creador y cuenta con dos atributos, donde uno de ellos es un literal no tipado (su descripción corta), y otro es un tipo de datos entero. Por otro lado, otra entidad, correspondiente al libro en el que aparece, la referencia. Las raíces comunes de las *URIs* se pueden representar con prefijos para facilitar la legibilidad.

Otro de sus grandes potenciales es el hecho de que son fuentes de datos estructuradas, a diferencia de otras fuentes como el contenido de sitios web, que son datos semiestructurados, o el texto libre, que son datos no estructurados. Gracias a esto, es posible utilizar sobre ellos lenguajes de consulta basados en patrones de búsqueda, filtros y condiciones, como *SPARQL* [20], de la misma forma que se utiliza *SQL* en bases de datos tradicionales. Esto permite generar consultas cuyos resultados cumplen exactamente con los requisitos especificados, sin ambigüedades o falsos positivos, y a su vez aprovechar la estructura del grafo para orientar semánticamente las búsquedas.

Como gran inconveniente, debido a la naturaleza y estructuración de sus datos,

no es posible realizar de forma directa consultas de texto libre, como las utilizadas en los motores de búsqueda web. Esto, como se va a presentar a continuación, ha generado un todo un campo de investigación en constante desarrollo.

1.2. Búsquedas de texto libre en grafos de conocimiento

El acceso a la información contenida en grafos de conocimiento requiere el uso de lenguajes de consulta como *SPARQL*, algo que es complicado de cara a los usuarios finales. Dado que las consultas por palabras clave o texto libre son las más habituales y sencillas de utilizar, se han investigado y desarrollado dos principales aproximaciones para conseguir aplicarlas a los grafos de conocimiento.

La primera de ellas consiste en realizar una traducción directa de consultas textuales a *SPARQL*, ya sea de forma automática o semiautomática, solicitando en este último caso retroalimentación al usuario. Esto implica una pérdida inherente de precisión, debido a que puede no ser posible adaptar una consulta en texto libre a uno o varios modelos de consultas *SPARQL* predefinidos, sin realizar antes suposiciones sobre el objetivo de la misma. Debido a esto, su uso en grafos de conocimiento de propósito general puede ser complejo, y en muchos casos se deben adaptar manualmente a su estructura específica.

Por otro lado, existe la posibilidad de convertir las entidades del grafo a documentos, para que estos sean indexados por un sistema de recuperación de información tradicional. Esto se realiza "aplanando" las entidades a documentos virtuales, de tal forma que el documento correspondiente a una entidad estará compuesto por el contenido textual de sus atributos y de los objetos con los que se relaciona. Adicionalmente, los predicados pueden servir para agrupar los objetos, traduciéndose de forma directa a campos del índice de documentos. Esto permite asignarles pesos diferentes a la hora de realizar la búsqueda y a su vez agrupar dos o más predicados en un único campo, como se puede observar en la Figura 3.

Esta última aproximación implica un estudio detallado de la estructura del grafo y la estrategia de indexación, debido a que una búsqueda en texto libre puede producir falsos positivos muy fácilmente dada la naturaleza del grafo: una búsqueda por el nombre de una entidad, por ejemplo, podría pesar más otras entidades que simplemente se encuentren relacionadas con ella, debido al número de menciones en el texto. En grafos de propósito general de gran tamaño, la especificación de la estructura del índice y sus pesos puede ser una tarea inabordable de forma manual debido al elevado número de entidades y predicados posibles, y a la heterogeneidad de las relaciones que puede poseer cada entidad. Esto hace que establecer que un conjunto de predicados que conforman un campo del índice tenga mayor peso que otro sea una tarea aproximada y poco exhaustiva.

Para mitigar este problema, se han desarrollado aproximaciones para obtener dichos pesos mediante *Machine Learning*, creando así un nuevo campo de estudio denominado *Learning to Rank*[3]. Esto traslada la problemática a definir manualmente un conjunto de consultas y posibles documentos a devolver sobre las que entrenar los modelos, algo muy costoso al requerir realizar juicios de relevancia[4] sobre ellas por parte de múltiples personas. Así mismo, estos conjuntos pueden no ser representativos de las consultas reales que se van a realizar sobre él. Por último, la

[3] https://en.wikipedia.org/wiki/Learning_to_rank
[4] https://trec.nist.gov/data/reljudge_eng.html

exactitud y exhaustividad de los resultados juzgados, utilizados como *ground truth* para el entrenamiento y validación, puede a su vez ser ambigua, al realizarse con sistemas de recuperación de información existentes.

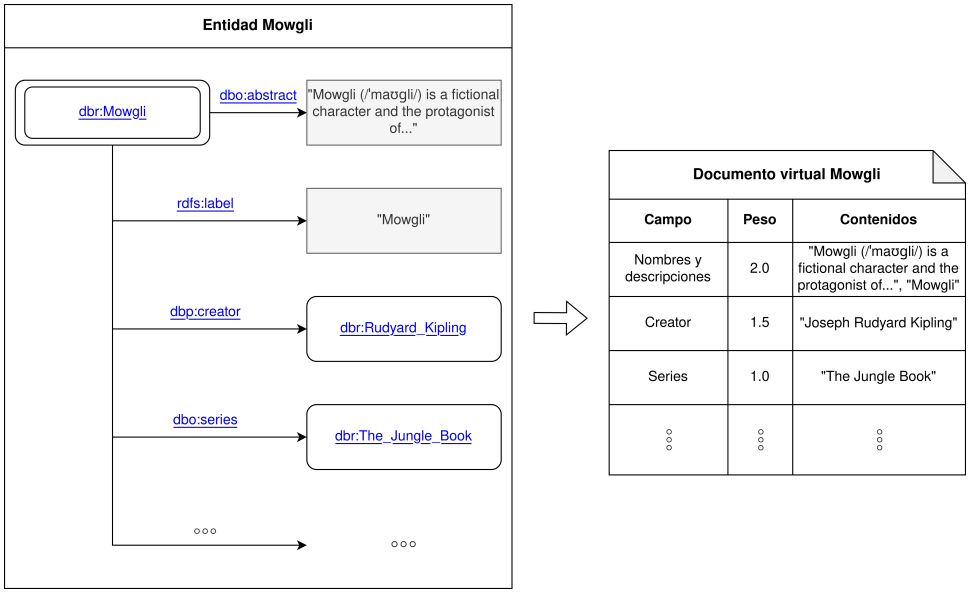

Figura 3: Ejemplo de conversión de las relaciones y atributos de una entidad en un grafo de conocimiento a un documento virtual, almacenable en un índice invertido. En este caso, se han agrupado los contenidos de los predicados de etiqueta y resumen en un mismo campo, con un peso mayor al resto. El texto de los objetos no literales (*URIs*) se ha extraído de sus atributos <*rdfs:label*>, que indican sus identificadores en lenguaje natural, o inferido a partir de las propias *URIs*.

1.3. Objetivos

El objetivo de este trabajo es desarrollar *Knowgly*, un sistema que realice la conversión de las entidades de un grafo de conocimiento a documentos virtuales, que serán alimentados al índice de un sistema de recuperación de información tradicional. Como novedad, se van a evitar las desventajas inherentes de dicha aproximación.

Para ello, vamos a aplicar una serie de métricas que indican la importancia, el grado de informatividad o la entropía, entre otras, de los sujetos, predicados y objetos que componen las tripletas del grafo a bajo nivel. Apoyándonos en dichas métricas, y partiendo de una división en el índice de los documentos en diferentes campos, que indica únicamente el número de ellos y sus pesos, obtendremos una organización óptima de los predicados a lo largo de dichos campos. Este esquema será utilizado en el sistema de recuperación de información final, tanto en la indexación como en las búsquedas.

De esta forma, el esquema se encontrará optimizado de forma automática teniendo en cuenta la estructura del grafo, sin requerir ningún conocimiento previo sobre el mismo ni ninguna acción manual durante todo el proceso.

1.4. Estructura de la memoria

En la Sección 2 se describen en detalle los conceptos básicos relativos al trabajo, introduciendo para ello los grafos de conocimiento y la recuperación de información semántica y orientada a entidades, así como la problemática actual.

En la Sección 3, se describen las herramientas y librerías utilizadas para la implementación del sistema desarrollado. Así mismo, se detallan el conjunto de datos utilizado para evaluar el sistema y el grafo de conocimiento utilizado como entrada.

La Sección 4 muestra la implementación y organización del sistema, detallando cada una de las etapas de ejecución del mismo.

La Sección 5 detalla los resultados obtenidos, ofreciendo una comparativa frente a los obtenidos por otros sistemas que utilizaban el mismo conjunto de datos de evaluación.

Finalmente, en la Sección 6 se recogen las conclusiones finales sobre el trabajo, así como una visión de posibles tareas a realizar como continuación del mismo.

2. Contexto tecnológico

2.1. Grafos de conocimiento

Los grafos de conocimiento, actualmente, son uno de los pilares fundamentales de la Web Semántica. Esto es debido a la tecnología de los datos enlazados (*Linked Data*[5]), que consiste en definir conceptos con ciertos atributos y relaciones, enlazándolos a su vez con conceptos pertenecientes a fuentes de datos externas. Esto consigue que, en conjunto, sean navegables e interpretables por máquinas.

2.1.1. Modelo básico

La estructura subyacente de las tecnologías de los datos enlazados se encuentra estandarizada bajo el modelo de datos *RDF* [18], el cual define una estructura de grafo mediante tripletas $<s, p, o>$, de la misma forma que se ha descrito anteriormente. De esta forma, *RDF* nos permite definir los grafos de conocimiento.

Este modelo básico, que únicamente define el modelo de datos a bajo nivel, es a su vez extendido por los estándares *RDFS* [19] y *OWL* [17], que definen un conjunto de relaciones a utilizar para definir ontologías. Estas ontologías, en esencia, definen un conjunto de relaciones, una jerarquía de clases a asignar a entidades y restricciones lógicas sobre todo ello.

Jerarquías de clases Las clases, también denominadas conceptos o tipos, establecen una jerarquización de las entidades, pudiendo referirse a cualquier conjunto de conceptos organizables en una taxonomía, como las categorías de productos en un comercio web o especies de seres vivos.

De este modo, es posible agrupar entidades bajo un mismo significado semántico, como *<dbr:Michael_Schumacher>* y *<dbr:Lewis_Hamilton>* bajo la clase *<dbo:RacingDriver>*, una clase específica para pilotos de carreras, así como bajo *<dbo:Person>*, una clase más general que engloba a todas las entidades que sean seres humanos. Con el objetivo de permitir la creación de jerarquías, es posible definir subclases con varios niveles de profundidad, como se puede observar en la Figura 4. Todas las clases están representadas como una entidad en el grafo, es decir, una *URI*, con la cual el resto de entidades se relacionan mediante un predicado de relación de tipo específico, *<rdf:type>*.

Nuestro sistema usa de forma extensiva las jerarquías de clases que poseen los grafos, basando muchas de las métricas utilizadas en ellas, debido a la capacidad que ofrecen para agrupar semánticamente los elementos del grafo.

Restricciones sobre relaciones Las relaciones controladas por una ontología y sus restricciones acotan las posibles formas en las que puede enlazarse, mediante predicados, una entidad de una clase con otra, así como los atributos que podemos asociar a cada una.

Para ello, establecen dos categorías de predicados: las *Object Properties* y las *Datatype Properties*. Por un lado, las *Object Properties* son todos los predicados

[5] https://en.wikipedia.org/wiki/Linked_data

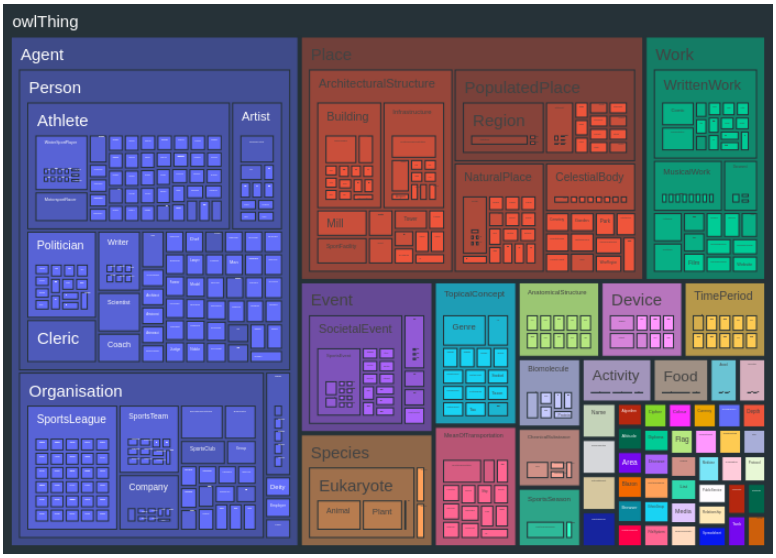

Figura 4: Diagrama de la jerarquía de clases de la ontología de la *DBpedia*, en el que se puede observar la herencia entre diferentes grupos de clases. Se puede observar cómo todas ellas son subclases de *owl:Thing*, estableciendo así una clase común a todas las instancias.

Fuente: https://www.dbpedia.org/resources/ontology/.

del tipo <*owl:ObjectProperty*>, al cual se le ha aplicado la restricción de que dichos predicados solo puedan encontrarse asociados con objetos no literales, es decir, *URIs*. Por otro lado, las *Datatype Properties* son aquellos predicados del tipo <*owl:DatatypeProperty*>, que en este caso marca que solo puedan encontrarse asociados con objetos literales.

Estas restricciones permiten, como en el caso anterior, jerarquizar los predicados del mismo modo que con las entidades. Así mismo, existen muchas otras restricciones, como las de cardinalidad o transitividad, que dictan la naturaleza de las relaciones e incluso abren la posibilidad de inferir relaciones implícitas. En el caso de *Knowgly*, hemos utilizado la distinción entre *Object Properties* y *Datatype Properties* durante el cálculo y uso de las métricas.

2.1.2. Potencial de los grafos de conocimiento

Gracias al control ofrecido por las ontologías, es posible definir una conceptualización arbitraria sobre cualquier aspecto del mundo real, de la forma más acotada, y por tanto precisa, que se desee. Esta capacidad, combinada con la interpretabilidad del grafo por máquinas, permite que sea posible verificar que las entidades y relaciones que utilizan una ontología dada no incumplan ninguna restricción, e incluso inferir nuevos conocimientos, es decir, relaciones definidas implícitamente.

Todo esto, en definitiva, permite a las máquinas acceder y explotar un conjunto de conocimientos reales de forma estructurada, y es lo que ha llevado a multitud de organizaciones a utilizarlas para diversos fines.

2.1.3. Uso actual

Actualmente, el grafo de conocimiento abierto más importante, tanto en relevancia general como en tamaño, es el definido por la *DBpedia*[6]. Este grafo es poblado con información extraída de la *Wikipedia*, de tal forma que se extraen las entidades sobre las que versa cada artículo, generando atributos y relaciones a partir de ellos. Así mismo, cuenta con una jerarquía de clases extensiva, basada y enriquecida a partir las categorías asignadas a dichos artículos.

Este grafo es creado de forma semi-automática, dependiendo en gran medida de la calidad y estructuración de la información de la *Wikipedia*, con una gran multitud de tratamientos que mejoran su calidad cada año. Toda su información contenida es ofrecida de forma abierta para su uso general, ofreciendo puntos de acceso para consultas *SPARQL*, interfaces web y volcados del grafo en ficheros de tripletas.

Por otro lado, existen muchos otros grafos de conocimiento general abiertos, como *YAGO*[7], específicos a áreas de conocimiento específicas, como *MeSH*[8], o cerrados, que son los grafos de conocimiento utilizados internamente por *Google*, *Bing*, *Diffbot*[9] o *Facebook*, entre otros.

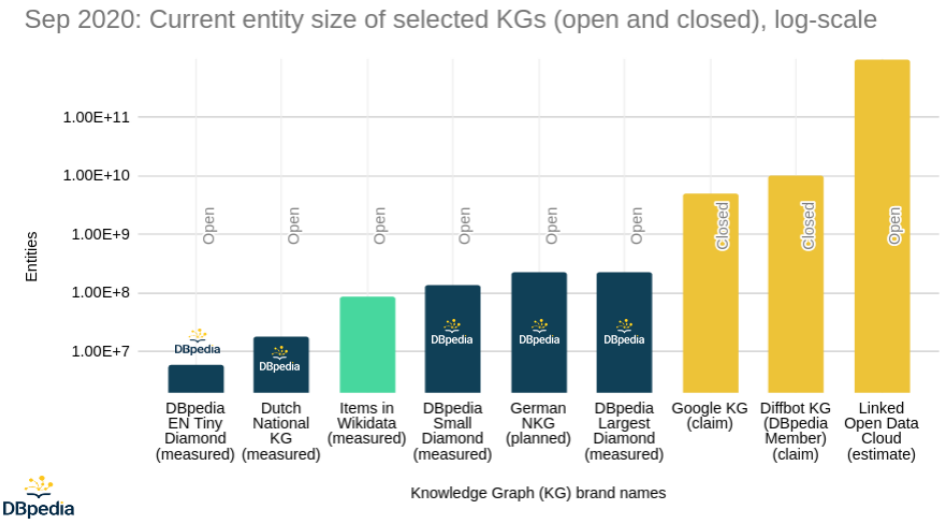

Figura 5: Estimaciones, a fecha de Septiembre de 2020, del tamaño de los mayores grafos de conocimiento, abiertos y cerrados, en número de entidades (nótese que no es el número de tripletas, que es mucho mayor). En conjunto, existen más de cien mil millones de entidades accesibles públicamente.

Fuente: https://www.dbpedia.org/resources/knowledge-graphs

[6] https://www.dbpedia.org/
[7] https://yago-knowledge.org/
[8] https://www.ncbi.nlm.nih.gov/mesh
[9] https://www.diffbot.com/

2.2. Recuperación de información semántica

Aunque no existe una definición consensuada de la recuperación de información semántica (*Semantic Search*), tal y como describe Krisztian Balog [1], conceptualmente engloba un amplio conjunto de métodos y técnicas que mejoran el acceso a la información a los usuarios, teniendo para ello en cuenta el contexto de su actividad y su intención. Esto mismo, trasladado a una búsqueda de cualquier tipo en un sistema de recuperación de información, implica tener en cuenta la semántica implícita de la consulta realizada. Esto significa, por ejemplo, analizar las dependencias entre términos o asociar la consulta a una clase específica de una ontología.

Con el auge de la Web Semántica, la recuperación de información semántica ha ganado notoriedad debido a la proliferación de los grafos de conocimiento, que, como se ha descrito anteriormente, permiten tener en cuenta dicha semántica implícita de las consultas. Esto ha provocado que estas técnicas se asocien directamente con los grafos de conocimiento.

2.3. Búsqueda orientada a entidades

La búsqueda orientada a entidades es un paradigma de búsqueda que consiste en organizar y acceder a la información en torno a las entidades de un grafo de conocimiento, a través de sus relaciones y atributos.

Como se ha detallado anteriormente, el concepto de entidad se puede observar desde dos puntos de vista: desde el del usuario y desde el técnico. Desde el primero, una entidad representa cualquier concepto del mundo real, como personas u organizaciones. Por otro lado, desde el punto de vista técnico, una entidad se puede asociar directamente a un sujeto s de un grafo de conocimiento dado, cuyas relaciones son todos los pares $< p, o >$ para los cuales existe una tripleta $< s, p, o >$, siendo o una *URI*. De la misma manera, los atributos serán todos esos pares $< p, o >$ para los cuales o sea un objeto literal.

Las entidades son mucho más versátiles que un documento clásico, debido a que, a partir de una de ellas, se puede realizar un recorrido por el grafo de conocimiento para cualquier finalidad. Esto permite que puedan utilizarse tanto como objetivo final (buscar n entidades y permitir al usuario acceder a más información a partir de ellas) y como herramienta intermedia para orientar y mejorar búsquedas en repositorios de información clásicos, con técnicas como el *Entity linking*[10], cuyo ejemplo se puede observar en la Figura 6.

2.4. Uso de métricas en grafos de conocimiento

Uno de los requisitos más comunes en tareas que hacen uso de grafos de conocimiento es la necesidad de inferir la importancia de ciertos elementos de un grafo cualquiera, como las entidades o los predicados del mismo, frente al resto de ellos, siguiendo cierto criterio. Esto se realiza aplicando un conjunto de métricas al grafo, que consisten en funciones que establecen recorridos a lo largo de él y devuelven puntuaciones numéricas para cada elemento a evaluar.

En este trabajo, vamos a utilizar y evaluar dos familias diferentes de métricas, con el objetivo de apoyarnos en ellas para inferir la importancia de cada uno de los

[10] https://en.wikipedia.org/wiki/Entity_linking

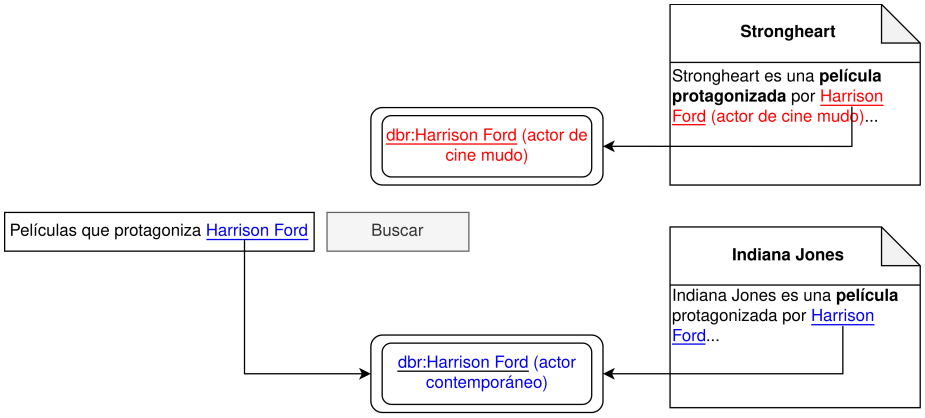

Figura 6: Ejemplo de *Entity Linking*, técnica en la que se anotan documentos y consultas mediante entidades, con el objetivo en este caso de desambiguar y orientar mejor las consultas. En este ejemplo, el usuario intenta buscar películas protagonizadas por Harrison Ford, cuya consulta se ha anotado manual o automáticamente para asociarle su entidad correspondiente. Al conocer a qué entidad se refiere cada documento de la colección, el buscador es capaz de evitar devolver documentos asociados a entidades ajenas, que con técnicas puramente sintácticas habrían provocado un falso positivo.

predicados del grafo bajo diferentes criterios. De esta forma, es posible obtener una organización óptima de los predicados a lo largo de los diferentes campos del índice a utilizar en el sistema de recuperación de información final.

2.4.1. Métricas de importancia estructural

Las métricas de importancia estructural provienen de una implementación de un sistema de generación de *infoboxes* y descripciones textuales de entidades, como las utilizadas en los motores de búsqueda web tradicionales. Para esta tarea, Hasibi et al. [5] requerían inferir los mejores hechos o *facts* de una entidad concreta, como pueden ser la fecha de nacimiento, el estilo de música o los premios recibidos para un músico famoso.

En el ámbito de un grafo de conocimiento esta tarea corresponde a, dada una entidad s, obtener los $facts < p, o > \in < s, p, o >$ más relevantes, teniendo en cuenta los tipos asociados a s dentro de la jerarquía. Para conseguir inferir los mejores *facts*, se diseñó un conjunto de métricas que dictan sus importancias en función de cada tipo de la jerarquía, siguiendo los siguientes criterios heurísticos:

- **Importancia**: La importancia de un *fact* f refleja la importancia general de f a la hora de describir una entidad s, independientemente de la necesidad de información subyacente (es decir, la mejor descripción general posible).
- **Relevancia**: La relevancia de un *fact* f refleja la cantidad de información que proporciona sobre una entidad s frente a una necesidad de información concreta.
- **Utilidad**: La utilidad de un *fact* f es la combinación de la importancia y relevancia de dicho *fact* frente a una necesidad de información, siendo simplemente

la suma de ambas puntuaciones. De esta forma, f tendrá una componente de importancia global y otra específica a la consulta.

Para conseguir una ordenación según estos criterios, se definió un conjunto de funciones, en las cuales o bien se realizan conteos de frecuencias de aparición de tripletas en el grafo, en función de ciertas restricciones, o bien se combinan y ponderan diferentes conteos y métricas intermedias. Éstas métricas son las siguientes:

- $\mathbf{FF(f)}$ (*Fact frequency*): Frecuencia del fact f en el grafo (considerado como *Knowledge Base* en la publicación original, es decir, KB).

$$|\{< s,p,o > \mid < s,p,o >\in KB,\ p = f_p,\ o = f_o\}|$$

  donde:

$$f =< p,o >:\ f_p = p,\ f_o = o$$

- $\mathbf{FF_p(p)}$ (*Fact frequency of predicate*): Número de *facts* que contienen al predicado p.

$$|\{< s',p',o' > \mid < s',p',o' >\in KB,\ p = f_p\}|$$

- $\mathbf{FF_o(o)}$ (*Fact frequency of object*): Número de *facts* que contienen al objeto o.

$$|\{< s',p',o' > \mid < s',p',o' >\in KB,\ o = f_o\}|$$

- $\mathbf{EF(f)}$ (*Entity frequency of fact*): Número de entidades diferentes que aparecen para un *fact* f dado.

$$|\{e \mid e \in \mathcal{E},\ f \in \mathcal{F}_e\}|$$

  donde:

$$\mathcal{F}_e : \{< p,o > \mid < s,p,o >\in KB,\ s = e\}|$$

- $\mathbf{EF_p(p)}$ (*Entity frequency of predicate*): Número de entidades únicas que contienen una relación con el predicado p.

$$|\{e \mid e \in \mathcal{E},\ \exists f \in \mathcal{F}_e : f_p = p\}|$$

- $\mathbf{EF_o(o)}$ (*Entity frequency of object*): Número de entidades únicas que contienen una relación con el objeto o.

$$|\{e \mid e \in \mathcal{E},\ \exists f \in \mathcal{F}_e : f_o = o\}|$$

- $\mathbf{EF_p(p,t)}$ (*Entity frequency of predicate for a given type*): Número de entidades de clase o tipo t que contienen una relación con el predicado p.

$$|\{e \mid e \in \mathcal{E},\ t \in tipos(e),\ \exists f \in \mathcal{F}_e : f_p = p\}|$$

- $\mathbf{TF_p(p)}$ (*Type frequency of predicate*): Número de clases o tipos diferentes de entre los de cada entidad que contiene una relación con el predicado p.

$$|\{t \mid < s,p',o >\in KB,\ p = p',\ t \in tipos(s)\}|$$

- **TImp$_\mathbf{p}$(p, t)** (*Type-based importance*): Ponderación entre $EF_p(p,t)$, el número de tipos en el grafo y $TF_p(p)$, que indica la importancia de un predicado según un tipo dado. Esta ponderación consigue pesar más aquellos predicados que están asociados a menos tipos, y por tanto pueden ser más relevantes para una entidad que otros que se encuentren asociados a entidades de tipos muy variados.

$$TImp_p(p,t) = EF_p(p,t) \cdot log\left(\frac{|\mathcal{T}|}{TF_p(p)}\right)$$

  donde:

$$\mathcal{T} = \{o \mid < s,p,o > \in KB,\ p =< \text{rdf:type} >\}$$

Para la tarea original, se establecía un conjunto de medidas en función de las anteriores, utilizadas para identificar predicados poco comunes con objetos frecuentes y viceversa, pesando para ello conteos de frecuencias. Así mismo, establecían métricas destinadas a medir la similaridad semántica de un *fact* frente a una consulta textual. Todas estas métricas finales se alimentaban a un sistema de *Machine Learning*, de tal forma que obtenían un sistema de inferencia de la importancia de *facts* de una entidad frente a una consulta dada.

En el caso de *Knowgly*, requerimos simplemente ordenar los predicados en función de un criterio de importancia global. Para conseguir esto, hemos expandido la métrica *Type-based importance* existente con dos nuevas familias de métricas desarrolladas por Carlos Bobed Lisbona, que se esperan publicar próximamente.

El efecto final de estas nuevas métricas es conseguir mejorar *Type-based importance*, que es una medida global al grafo que indica la importancia del predicado p respecto a un tipo t en función de frecuencias de entidades. Al ponderarla con la primera familia de métricas, destinadas a medir la entropía de los objetos de cada predicado, se valoran más los predicados con mayor probabilidad de tener objetos variados. Así mismo, al pesarla a su vez por la segunda de ellas, se consigue penalizar tipos demasiado comunes.

La ponderación final de *Type-based importance* por ambas familias de métricas es la usada para inferir la importancia de un predicado frente al resto, que permite obtener una medida global para todos los tipos asociados a él o para un subconjunto de ellos.

2.4.2. Métricas de InfoRank

Las métricas de *InfoRank* provienen de *QUIRA* [10], un sistema de búsqueda de entidades basado en la generación automática de consultas *SPARQL*. En este sistema, se buscaba obtener los mejores predicados a utilizar en dichas consultas, basándose en cuales son más informativos y por tanto tienen mayor probabilidad de contener términos de una consulta de palabras clave. Del mismo modo, se buscaba ordenar las entidades devueltas según la cantidad de información que éstas proporcionan y el número de relaciones entrantes y salientes con el resto de ellas, pudiendo así establecer un criterio de *ranking* en *SPARQL*.

En este caso, los criterios heurísticos utilizados son los siguientes:

- **Los nodos importantes poseen mucha información sobre ellos**: En un grafo de conocimiento, esto se traduce a que las entidades más importantes

tienden a poseer un mayor número de relaciones con objetos literales, es decir, objetos cuyos valores no son una *URI*, que corresponde a una mayor calidad y exhaustividad de sus descripciones. Esto se puede observar, por ejemplo, con el número de datos que puede poseer la entidad de una película como *Titanic* frente a otras menos populares.

- **Los nodos importantes están rodeados de otros nodos importantes**: En un grafo de conocimiento, esto se traduce a que las entidades importantes, siguiendo el criterio anterior, tienden a poseer relaciones con otras entidades de importancia similar. Utilizando el mismo ejemplo, las entidades de las películas más populares tienden a relacionarse con actores famosos, que, a su vez, están mejor descritos que otros.
- **Tener pocos amigos es mejor que tener muchos conocidos**: Esto corresponde a la intuición de que es preferible una entidad relacionada con unas pocas entidades muy bien descritas frente a otra relacionada con muchas entidades poco descritas.

Para conseguir métricas en función del primer criterio, se define un conjunto de métricas similares a las de importancia estructural, con el objetivo de medir la informatividad de predicados y sujetos. En este caso, todas las medidas se pesan exclusivamente con conteos de objetos en relaciones bajo diferentes ponderaciones, sin distinguir tipos. Por otro lado, para conseguir el segundo criterio se utiliza *PageRank*[11] aplicado a grafos de conocimiento, que, para cada entidad, valora el número de sus relaciones y la calidad de las mismas. Para aplicar el último criterio, simplemente se ponderan las puntuaciones de *PageRank* por los criterios de informatividad anteriores, realizando para ello una variante de *PageRank* en la que se pesa cada relación de manera individual (*Weighted PageRank*).

Las métricas son las siguientes:

- **IW(s)** (*Informativeness of a resource s*): Número de tripletas $< s, p, o >$ en las que aparece s como sujeto y o es un literal.

$$|\{< s, p, o > \mid < s, p, o > \in KB,\ o \in \mathcal{L}\}|$$

  donde:

$$\mathcal{L} = \{o \mid < s, p, o > \in KB,\ \text{o no es una } URI\}$$

- **IR(p)** (object properties) (*InfoRank of an object property p*): Máximo valor de $IW(s) + IW(r)$ encontrado a lo largo de las tripletas $< s, p, r >$ del grafo. Esta medida es aplicable únicamente a *Object Properties*, es decir, a predicados asociados únicamente con *URIs* en sus objetos.

$$|\{max(IW(s) + IW(r)) \mid < s, p, r > \in KB\}|$$

- **W(s, p)** (*Peso normalizado para s y p*): Peso del predicado p para el sujeto s, que es la ponderación de $IR(p)$ por el sumatorio de $IR(q)$ para los predicados q asociados a s, bidireccionalmente.

[11] https://en.wikipedia.org/wiki/PageRank

Este último matiz es importante, ya que se tienen también en cuenta los pesos de predicados para todas las entidades relacionadas con s. Esto provoca que la medida sea local a cada entidad, teniendo en cuenta los pesos de los predicados respecto a sus relaciones.

$$W(s,p) = \frac{IR(p)}{\sum_{q \in P \text{ and } (<s,q,r> \in KB \text{ or } <r,q,s> \in KB)} IR(q)}$$

donde:

$$P = \{p \mid <s,p,o> \in KB\}$$

- $\mathbf{PR_W(s,i)}$ (*PageRank of an entity s*): *PageRank* bidireccional para una entidad s en la iteración i, pesando cada relación por $W(s,p)$.

$$PR_W(s,i) = \frac{1-\alpha}{N} + \alpha \sum_{<s,p,r> \in KB \text{ or } <r,p,s> \in KB} PR_W(r, i-1) \cdot W(s,p)$$

- $\mathbf{IR(s)}$ (*InfoRank of an entity s*): $PR_W(s,i)$ pesado por la informatividad de s, es decir, $IW(s)$.

$$IR(s) = PR_W(s,i) \cdot IW(s)$$

El *PageRank* calculado para cada entidad consiste en una adaptación directa de dicho algoritmo, comúnmente usado en el entorno de indexación web, a grafos de conocimiento. Esta adaptación consiste en tomar cada posible sujeto s en el grafo como una página del algoritmo original, mientras que los *links* salientes son las relaciones $< s,p,o >$, en las cuales s actúa como sujeto, y los *links* entrantes son las relaciones $< r,p,s >$, es decir, aquellas en las que s actúa como objeto.

El uso de *PageRank* en grafos de conocimiento tiene el problema inherente de que un sujeto s puede no ser relevante aunque la mayoría de las entidades lo enlacen, como puede ser el caso de las entidades que representan clases. Para remediar esto, se pesa cada relación en función de su informatividad mediante $W(r,p)$, y a su vez el resultado final es pesado por la informatividad de la entidad, $IW(r)$, obteniendo así el *InfoRank* de cada entidad.

En *Knowgly*, hemos aplicado una extensión de las de métricas publicadas originalmente en colaboración con la autora original, de tal forma que es posible aplicar $IR(p)$ para inferir la informatividad tanto de las *Object Properties* y *Datatype Properties*, donde estas últimas son los predicados únicamente asociados a objetos literales. Por otro lado, esta extensión permite aplicar la medida $W(r,p)$, y por tanto pesar el *PageRank*, en función de predicados de ambas clases.

Además de permitirnos inferir la informatividad de los predicados, el *InfoRank* resultante de cada entidad se ha aprovechado como medida de *reranking*, es decir, de reordenación de los resultados de una consulta, tal y como se hacía en *QUIRA*.

3. Herramientas utilizadas

Knowgly se ha programado principalmente en *Java*, debido a que es el lenguaje que cuenta con el ecosistema de librerías relacionadas con *RDF* más extenso. Estas librerías, por tanto, se han utilizado para facilitar la interacción con grafos de conocimiento, tanto en alto como en bajo nivel.

Por otro lado, también hemos hecho uso de un sistema de recuperación de información clásico, sobre el que se han aplicado los esquemas de índice generados, y de un conjunto de datos de evaluación, utilizado para medir la eficacia del sistema, que se detallan a continuación.

3.1. Jena

Jena[12] es una librería utilizada para construir aplicaciones relacionadas con la Web Semántica y los datos enlazados, ofreciendo para ello un gran número de abstracciones para trabajar con tripletas *RDF* en alto nivel. Así mismo, ofrece la capacidad de generar, leer y operar sobre grafos de conocimiento, tanto de manera remota como local, desde ficheros o repositorios desarrollados por la propia librería, mediante consultas *SPARQL*.

Esta librería va a utilizarse como una abstracción para operar sobre grafos de conocimiento a alto nivel, para así realizar operaciones poco exigentes en requisitos de tiempo sobre ellos.

3.2. HDT

HDT[13] es una librería especializada en un formato específico, con el mismo nombre, para contenido *RDF*. Al contrario que los ficheros u otros repositorios específicos que permite utilizar *Jena*, el formato de *HDT* utiliza un único fichero en el cual se almacena el grafo como un diccionario binario de solo lectura. Este diccionario está diseñado para reducir al máximo el espacio utilizado y permitir realizar búsquedas únicamente con patrones de tripleta, que son búsquedas individuales de tripletas $< s, p, o >$ con valores específicos o desconocidos en cada posición, extremadamente eficientes en tiempo.

Esta librería, al contrario que *Jena*, ofrece únicamente dichas búsquedas por patrones de tripleta y un conjunto mínimo de operaciones a bajo nivel sobre el diccionario binario. Esta posibilidad de operar al nivel más bajo posible se aprovechará a la hora de realizar operaciones exigentes en términos de tiempo y uso de memoria, como durante la generación de métricas.

No obstante, ofrece una capa de integración sobre *Jena*, que permite a dicha librería operar sobre un fichero *HDT* internamente. Esto se ha aprovechado para poder realizar consultas *SPARQL* sobre el mismo, en situaciones en las que no haya requisitos exigentes de uso de recursos o tiempo.

[12] https://jena.apache.org/
[13] https://www.rdfhdt.org/

3.3. ElasticSearch

ElasticSearch[14] es el sistema de recuperación de información tradicional sobre el que se va a implementar el índice a generar. Este sistema utiliza *Lucene* internamente, que es actualmente uno de los sistemas de recuperación de información más eficientes y con mayor número de funcionalidades. *ElasticSearch* se ha elegido por el hecho de que ofrece un elevado número de abstracciones útiles sobre *Lucene*, que permiten ajustar la configuración de los índices de manera muy sencilla.

Cabe mencionar que *Knowgly* es independiente del sistema de recuperación de información final subyacente, ya que el único requisito es que utilice índices invertidos[15], cuya estructura, en alto nivel, es la que se encarga de generar nuestro sistema.

3.4. DBpedia-Entity

DBpedia-Entity [7] es una colección de pruebas diseñada para la búsqueda orientada a entidades, que es a lo que se dedica concretamente nuestro sistema. Esta consta de un conjunto de evaluación y un grafo de conocimiento a utilizar para ello.

Conjunto de evaluación El conjunto de evaluación contiene una serie de consultas, extraídas a su vez de diferentes colecciones de consultas específicas, utilizadas para evaluar sistemas de recuperación de información en general, que son las siguientes:

- ***SemSearch_ES***: Consultas de palabras clave que indican el nombre de una entidad concreta, a obtener idealmente como primer resultado. Ejemplos: "*joan of arc*", "*NAACP Image Awards*".
- ***INEX-LD***: Consultas clásicas de recuperación de información de palabras clave, que pueden referirse a una o varias entidades explícita o implícitamente. Ejemplos: "*vietnamese food blog*", "*baseball player most homeruns national league*".
- ***QALD2***: Consultas clásicas de recuperación de información similares a las de *INEX-LD*, pero en este caso escritas en lenguaje natural. Ejemplos: "*Which German cities have more than 250000 inhabitants?*", "*Who is the husband of Amanda Palmer?*".
- ***ListSearch***: Consultas clásicas de recuperación de información, escritas en palabras clave o lenguaje natural, que buscan específicamente una lista de entidades. Ejemplos: "*republics of the former Yugoslavia*", "*Airlines that currently use Boeing 747 planes*".

Dado este conjunto de consultas, los autores de *DBpedia-Entity* han recopilado juicios de relevancia mediante *crowdsourcing*[16], indicando de esta forma posibles entidades con alta, media, poca o ninguna relevancia para cada una de las consultas.

[14] https://www.elastic.co/
[15] https://en.wikipedia.org/wiki/Inverted_index
[16] https://en.wikipedia.org/wiki/Crowdsourcing

A partir de estos juicios, han indicado los resultados de dos sistemas de recuperación de información distintos, que utilizan también la misma técnica de transformación del grafo a un índice tradicional. Estos, a su vez, implementan diferentes modelos de recuperación, desde los modelos clásicos de *BM25* y *BM25F* [15] a modelos avanzados como *FSDM* [23] o *MLM* [12], que requieren aplicar *reranking* y ajustar un gran número de parámetros, ya sea manualmente o mediante técnicas de *Machine Learning*. Estos modelos, así como las técnicas utilizadas por ambos sistemas, serán detallados en profundidad en la Sección 5.1.

Todos estos resultados se han evaluado utilizando las medidas $NDCG@10$ y $NDCG@100$. $NDCG$[17] es una medida de evaluación utilizada para evaluar sistemas de recuperación web, ya que valora con mayores puntuaciones aquellos sistemas que devuelvan los documentos de mayor relevancia en los primeros n resultados, que en este caso son 10 y 100.

Conjunto de datos El grafo de conocimiento utilizado como base para la evaluación es el grafo de la *DBpedia* a fecha de Octubre de 2015[18], que consta de 212.737.087 tripletas, con 19.781.237 sujetos y 61.828 predicados únicos, así como 385 tipos en su jerarquía. Este grafo se ha utilizado en todo momento tanto para evaluar el comportamiento de *Knowgly* frente a grafos de gran tamaño como para comparar su eficacia, respecto a los resultados ofrecidos por la colección de pruebas.

Este grafo, a su vez, sirve como ejemplo perfecto para comprobar el comportamiento de *Knowgly* ante datos espurios, debido a su gran magnitud y al hecho de que es creado de forma semiautomática mediante *crawlers* que extraen información de los artículos de la *Wikipedia*. Esto implica que hay un elevado número de entidades y predicados espurios, que o bien referencian a un mismo concepto o cuyas *URIs* contienen faltas de ortografía. Por ejemplo, existen predicados *<dbp:familia>* y *<dbp:family>* refiriéndose al mismo concepto sin ningún tipo de relación entre ellos, o predicados extraídos incorrectamente como *<dbp:birthPlaxe>* y *<dbp:birthPalce>*.

Esto nos ha permitido evaluar el comportamiento del sistema, y, especialmente, de las métricas, ante grafos de cualquier tipo, que no necesariamente tienen que ser perfectos o incluso cumplir su ontología subyacente.

[17] https://en.wikipedia.org/wiki/Discounted_cumulative_gain
[18] http://downloads.dbpedia.org/wiki-archive/Downloads2015-10.html

4. Implementación y funcionamiento

4.1. Estructura general del sistema

Knowgly cuenta con una estructura de módulos separados por capas, desacopladas al máximo unas de otras, y es posible utilizarlo como ejecutable o como librería, permitiendo así que pueda ser integrado en otros sistemas. Así mismo, con el objetivo de facilitar la experimentación con él, cada uno de los módulos es configurable de forma extensiva.

Cada una de las capas implementa una etapa en la que se realizan tareas independientes del resto, partiendo de un grafo de conocimiento hasta llegar a un índice de documentos en el sistema de recuperación de información destino, asemejándose a la estructura clásica de los indexadores web. Esto facilita la organización y, por tanto, la comprensión del mismo, algo que se ha buscado en todo momento para que sea posible ampliarlo o utilizarlo como base para trabajos futuros. Las etapas definidas son las siguientes:

- **Etapa de aplicación de métricas** En esta etapa inicial se generan las métricas de importancia estructural y de *InfoRank* para el grafo de conocimiento de entrada, almacenándolas en un subconjunto de métricas junto al grafo original. Este proceso se detalla en la Sección 4.2.
- **Etapa de extracción e indexación de entidades** Partiendo del subconjunto de métricas generado anteriormente, se definen uno o varios agregadores de métricas que, consultándolas, definen un esquema del índice a seguir. Utilizando dicho esquema y un extractor de información de entidades del grafo de conocimiento, el indexador genera documentos virtuales para cada entidad y los introduce en el sistema de recuperación de información final, de la misma forma que se describió en la Figura 3. Este proceso se detalla en la Sección 4.3.
- **Etapa de búsqueda** Utilizando la configuración del indexador, que indica el número de campos en el índice generado, se instancia un buscador que lanza las consultas sobre el sistema de recuperación de información y devuelve los resultados, permitiendo hacer funciones adicionales como el *reranking* de los mismos. Esta etapa se detalla en la Sección 4.4.
- **Etapa de evaluación** A partir de un conjunto de consultas de evaluación, ejecuta las etapas de extracción, indexación y búsqueda bajo distintas configuraciones, almacenando los resultados de las consultas. Esta última etapa es solo utilizada para evaluar el comportamiento del sistema ante diferentes configuraciones, y se detalla en la sección 4.5.

En la Figura 7 se puede observar la división completa de módulos respecto a las etapas del sistema, mostrando a su vez sus dependencias respecto a las fuentes de datos utilizadas. Por otro lado, en la Figura 8 se puede observar un ejemplo de las operaciones realizadas a lo largo de las etapas intermedias, que se van a desglosar a continuación, para conseguir indexar una entidad en el sistema final.

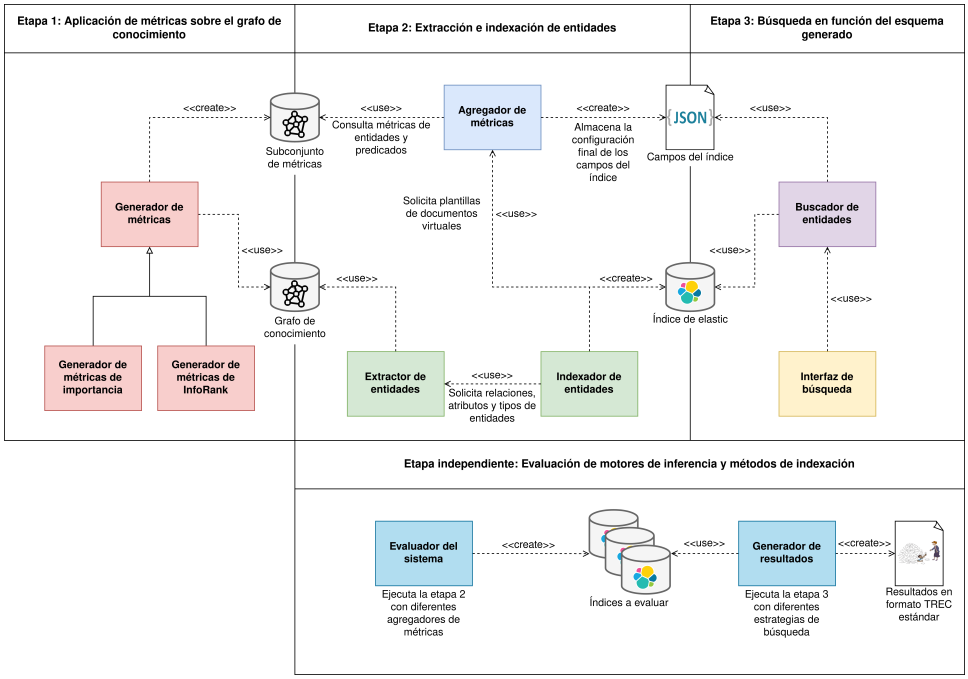

Figura 7: Separación de módulos y etapas, señalando las relaciones entre módulos y respecto a las fuentes de datos utilizadas en el sistema.

4.2. Etapa de aplicación de métricas

La etapa de aplicación de métricas consiste en generar un subconjunto de métricas a partir de un grafo de conocimiento de entrada, sin conocer ningún detalle previo ni realizar suposiciones sobre su estructura.

Todo los cálculos de métricas necesarios se realizan *offline* en este punto, de tal forma que las etapas posteriores únicamente deben leer las métricas almacenadas.

4.2.1. Generadores de métricas

El generador de métricas, que se encarga de aplicar ambas familias de métricas sobre el grafo de conocimiento, es el módulo con los requisitos de tiempo, memoria y espacio de almacenamiento más estrictos.

Inicialmente, realizamos una implementación básica de cada familia de métricas en alto nivel, mediante consultas de inserción de *SPARQL* a ejecutar secuencialmente por *Jena*. Esta estrategia solo ha podido aplicarse sobre grafos de pequeño tamaño, debido a que, a medida que crece el grafo, el tamaño del subconjunto de métricas y los requisitos de recursos durante su cálculo crecen cuadráticamente con el número de predicados y de objetos[19]. Debido a la sobrecarga adicional en tiempo y memoria derivada de la interpretación de las consultas *SPARQL*, aun tras

[19]La gran mayoría de submétricas de *InfoRank* e importancia estructural constan de un bucle principal, en función de los predicados del grafo, y una secuencia de bucles internos en función de objetos, tipos o resultados de otras submétricas. Al ejecutarse cada una secuencialmente, el coste global es cuadrático en ambos casos.

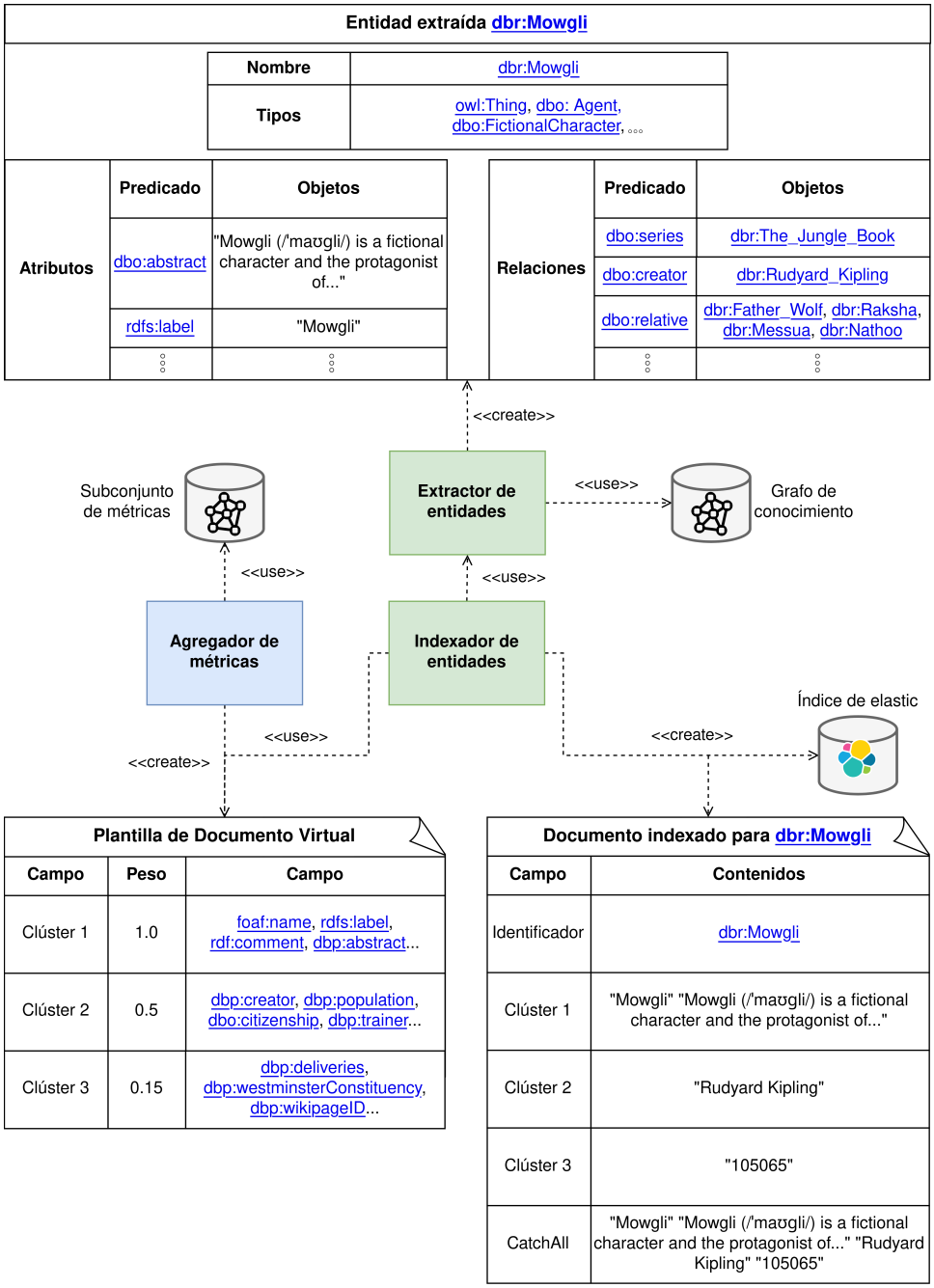

Figura 8: Ejemplo de indexación de la entidad <*dbr:Mowgli*>, simplificada para mostrar menos relaciones. En el lado izquierdo, el agregador genera la plantilla de documento virtual, que indica las correspondencias de predicados a campos. En la parte superior se encuentra la información extraída para la entidad, como las relaciones, atributos o tipos. En el lado derecho, se muestran los contenidos indexados en el sistema final, con las representaciones textuales de los objetos de las relaciones y atributos, distribuidas tal y como dicta la plantilla.

optimizarlas, resulta inadmisible su aplicación en conjuntos de datos de tamaño considerable, como el grafo de la *DBpedia* a utilizar.

Para contrarrestar este problema, se ha realizado una implementación a bajo nivel de todo el proceso de generación de métricas, utilizando exclusivamente el formato *HDT* y su librería. Para ello, se han reimplementado las consultas *SPARQL* programáticamente con operaciones básicas sobre su diccionario binario.

Esto permite llegar mucho más allá de lo que los motores de ejecución y optimización de consultas *SPARQL* son capaces de realizar, ofreciendo las siguientes ventajas:

- **Es posible paralelizar el cálculo de las métricas**: Esto es algo que se ha realizado extensivamente en todas las métricas intermedias y finales para paliar sus costes en tiempo de cálculo. Debido a que las métricas se calculan sobre predicados o entidades, con varios subniveles de agrupación bajo ellos, es posible realizar un cálculo paralelo de las métricas sobre cada elemento, sin ninguna contención entre hilos más allá del almacenamiento de los resultados.

- **Se puede controlar el uso de la memoria**: Gracias a que el formato *HDT* almacena las referencias a sujetos, predicados y objetos mediante índices numéricos, es posible almacenar resultados de métricas en memoria y dereferenciarlos a *strings* únicamente al escribir los resultados finales.

  Así mismo, realizando un análisis exhaustivo de las dependencias entre las métricas a calcular, es posible decidir qué conteos u otros aspectos se pueden almacenar en memoria como caches temporales, así como qué resultados intermedios desechar. Por el contrario, en la implementación de *SPARQL*, se debía almacenar todas las métricas intermedias en el grafo, incurriendo también en costes elevados de almacenamiento.

El resultado en ambos casos es un subconjunto de métricas que, según la configuración elegida, se añade como un subgrafo al grafo de conocimiento original, o como un fichero de *HDT* combinado con el del grafo. En la Figura 8, se puede observar que actúa como la fuente de datos para los agregadores de métricas.

4.2.2. Generador de PageRank

Tal y como requiere la implementación original de *InfoRank*, se ha realizado una implementación de *Weighted PageRank* bidireccional como parte del trabajo.

La ejecución del algoritmo de *PageRank* en grafos, del mismo modo que en la Web, es especialmente costosa a medida que crece el número de tripletas, y por tanto también se debe implementar a bajo nivel. Para ello, se ha adaptado una implementación altamente paralelizada basada en el método de *power iteration*, de tal forma que, en una iteración i, es posible calcular el *PageRank* de un sujeto independientemente del resto, siempre y cuando se almacenen los datos de la iteración inmediatamente anterior.

Partiendo de una implementación inicial de *PageRank* para *HDT* paralelizada por Carlos Bobed Lisbona[20], se han realizado las siguientes adaptaciones:

- **Se ha añadido un componente de bidireccionalidad**: La implementación de *PageRank* requerida por *InfoRank* requiere que, para una entidad s,

[20] https://github.com/cbobed/PageRankRDF

se tomen como entidades con *links* salientes hacia s todas aquellas relacionadas con s, independientemente de su dirección. Esto implica que todas las entidades r presentes en tripletas $< r, p, s >$ y $< s, p, r >$ aportan puntuación al *PageRank* de s. En una implementación unidireccional, solo aportarían puntuación las entidades r de tripletas $< r, p, s >$.

- **Se ha añadido pesado a nivel de relaciones**: El *Weighted PageRank* requiere que se pese cada una de las relaciones $< r, p, s >$ y $< s, p, r >$ en función de una puntuación del *link*, que en este caso es p. Para ello, se ha incluido el cálculo de la puntuación de $W(s, p)$, tal y como se especifica en *InfoRank*.

Adicionalmente, y con el objetivo de mantener la eficiencia de la implementación original, se ha incluido el cacheo de los valores de $W(s, p)$ calculados. Esto elimina la necesidad de realizar consultas y recalcular dichos valores a partir de la segunda iteración. Al mantener mapas de índices númericos de *HDT* en vez de *strings*, del mismo modo que en el cálculo de las métricas, el uso de memoria se ha minimizado.

4.3. Etapa de extracción e indexación de entidades

La etapa de extracción e indexación de entidades parte del grafo de entrada y el conjunto de submétricas generado anteriormente, de tal forma que se genera un esquema de índice mediante dichas métricas y se extraen e indexan todas las entidades del grafo sobre el sistema de recuperación de información final elegido.

4.3.1. Agregadores de métricas

Funcionamiento Los agregadores de métricas se encargan de generar el esquema del índice que seguirán el resto de módulos a partir de este punto, devolviendo como resultado plantillas de documentos virtuales a rellenar. Para ello, se apoyan en las métricas generadas anteriormente para inferir la importancia de cada uno de los predicados frente al resto, según diferentes criterios.

En todos los casos, su funcionamiento se basa en asignar una puntuación numérica a cada uno de los predicados del grafo, agrupándolos en función de ellas en n subconjuntos, mediante el algoritmo de *KMeans++*[21].

Una vez ejecutado el algoritmo, cada uno de los clústeres de predicados resultantes se traduce directamente a un campo del índice a utilizar, tal que todos los predicados del clúster son asignados a dicho campo. Estos clústeres, y por tanto los campos, son ordenados según los valores de sus centroides, consiguiendo así que, en la etapa de búsqueda, cada uno de los campos pueda ser pesado según su importancia. En la Figura 9 se puede observar un ejemplo real de clusterización.

Como funcionalidad adicional, los agregadores de métricas permiten definir combinaciones de distintos criterios de métricas, permitiendo así asignar a un predicado una combinación pesada de puntuaciones provenientes de distintas familias. Para ello, se aplica la media geométrica ponderada[22], donde $peso \in [0, 1]$, sobre cada

[21] https://en.wikipedia.org/wiki/K-means++
[22] https://en.wikipedia.org/wiki/Weighted_geometric_mean

Figura 9: Ejemplo de plantilla de documento virtual generada y almacenada en *JSON* por un agregador de métricas, usando exclusivamente métricas de *InfoRank*. Los clústeres se encuentran ordenados por importancia, y se puede observar que el clúster de mayor peso (*bucket0*) contiene predicados de nombre y tipo, que son muy relevantes. Se han omitido predicados para facilitar la visibilidad, debido a que los clústeres de mayor, media y menor prioridad contenían 347, 3982 y 57499 predicados, respectivamente.

predicado, que se puede observar en la Fórmula 1.

$$score(p,\ agregador_1,\ agregador_2,\ peso) = \\ score(p,\ agregador_1)^{peso}\ \cdot\ score(p,\ agregador_2)^{(1-peso)} \tag{1}$$

Esto nos permite combinar criterios de métricas de importancia estructural e *InfoRank* a la vez, independientemente de los rangos de valores para las puntuaciones de cada agregador.

Durante la experimentación, hemos podido observar que las clusterizaciones realizadas por el algoritmo *KMeans++*, que se ejecuta sin límite de iteraciones y repite 10 veces, devolviendo la mejor clusterización, son idénticas en ejecuciones sucesivas.

Debido a esto, podemos confirmar empíricamente que el algoritmo consigue converger con los criterios que imponemos. Así mismo, el coste en tiempo del algoritmo es negligible al ser datos unidimensionales.

Agregadores de métricas implementados Los agregadores de métricas definidos son los siguientes:

- **Sobre las métricas de importancia estructural:**
  - ***Predicate-Based Structural Importance Geometric Mean***: Media geométrica de las métricas desarrolladas como extensión de las métricas de importancia estructural.
  - ***Predicate-based Structural Importance Multiplication***: Multiplicación de las métricas desarrolladas como extensión de las métricas de importancia estructural (idénticas a la media geométrica, sin su exponente).
  - ***Predicate-based Structural Importance Harmonic Mean***: Media armónica de las métricas desarrolladas como extensión de las métricas de importancia estructural. Este último se ha excluido al generar clusterizaciones de baja calidad.
- **Sobre las métricas de *InfoRank***:
  - ***Predicate-based InfoRank***: Valor de $IR(p)$ para cada predicado, tras extender dicha métrica para poder aplicarla a *Datatype Properties*.

Adicionalmente, se han definido agregadores adicionales para poder construir plantillas de documento virtual a nivel de cada entidad, asignando pesos a los predicados asociados a ella en función de sus tipos, y por tanto optimizándose localmente a ellas. Esto se ha incluido como parte del trabajo futuro. En la Figura 8 se puede observar la creación de una plantilla de documento virtual.

4.3.2. Extractor de entidades

El extractor de entidades se encarga de extraer las relaciones y atributos asociados a un sujeto del grafo de conocimiento. Este se encuentra implementado como un módulo separado, con el objetivo de ofrecer una configuración extensiva de su comportamiento, que afecta considerablemente a la calidad de los resultados del sistema. Del mismo modo, permitimos así su extensión en el futuro, ya que no solo es aplicable a la tarea actual de indexación.

De esta forma, permite definir listas de predicados a evitar extraer, la distinción y extracción de elementos adicionales, como tipos de entidades, y múltiples comportamientos a la hora de generar las representaciones textuales de *URIs* y valores literales. Así mismo, permite realizar consultas en la vecindad de la entidad, permitiendo obtener dominios y rangos de predicados asociados a ella, que definen los tipos asociados de forma implícita a los sujetos y objetos de dicho predicado respectivamente, algo que se utilizará en el futuro.

En la Figura 8 se puede observar la información extraída para <*dbr:Mowgli*>. Su combinación junto a la plantilla de documento virtual, que indica las correspondencias de predicados a campos, es la que facilita la conversión de la entidad a un documento virtual.

4.3.3. Indexador de entidades

El indexador de entidades utiliza los dos módulos anteriores para generar documentos virtuales de cada entidad del grafo, que serán introducidos en el índice del sistema de recuperación de información destino.

Para ello, utiliza el extractor de entidades para obtener todas las relaciones y atributos de cada entidad, y asigna las representaciones textuales de los objetos con los que estaba relacionado cada predicado a su campo correspondiente del índice, que es indicado por el agregador de métricas.

Esta tarea, aunque no presenta problemas de uso excesivo de memoria como en el caso de los generadores de métricas, posee requisitos estrictos de tiempo. Esto es debido a que indexar cada entidad implica realizar las siguientes tareas:

1. Realizar un recorrido en el grafo a lo largo de sus relaciones.
2. Tratar las representaciones textuales de los objetos encontrados, realizando recorridos adicionales según la estrategia elegida, como el extraer etiquetas textuales de atributos.
3. Generar un documento virtual a alimentar al sistema final, que deberá añadir en su índice almacenado en disco.

Debido a que el número de entidades depende completamente del grafo, llegando a tener 22.268.571 entidades en el caso del grafo de la *DBpedia* utilizado, esto puede ser costoso en tiempo.

Para evitar estos problemas, hemos implementado múltiples optimizaciones. Desde el lado de la extracción, para evitar de nuevo la sobrecarga en tiempo derivada de la interpretación de consultas *SPARQL*, hemos implementado una versión que opera directamente sobre ficheros *HDT* para el extractor de entidades. Esto minimiza el tiempo derivado de los recorridos del grafo.

Por otro lado, el proceso de indexación ha sido paralelizado sobre las entidades, y se ha minimizado el coste derivado de operar sobre el disco mediante solicitudes de indexación por lotes.

El resultado final es un índice que consta de tantos campos como han indicado los agregadores de métricas, además de un campo *catchAll* que engloba los contenidos de todos ellos. Este último es un campo auxiliar, añadido para poder establecer comparativas frente a los índices de sistemas existentes, tal y como se detallará en la Sección 5. En la Figura 8 se puede encontrar el contenido del índice del sistema final para *<dbr:Mowgli>*, incluyendo el campo *catchAll*.

4.4. Etapa de búsqueda

La etapa de búsqueda parte del índice del sistema de información final creado anteriormente, de tal forma que ya es posible realizar consultas sobre el mismo y obtener entidades como resultado.

En este punto, su única dependencia frente al resto de etapas es la necesidad de obtener el esquema general del índice. Este únicamente debe indicar el número y los nombres de los campos que contiene, sin necesitar obtener los predicados asociados a cada uno, que pasa a ser transparente al sistema. Por ello, se puede

obtener o bien a partir de una plantilla de documento virtual ya almacenada o a partir de la configuración inicial del módulo de agregación de métricas, eliminando la necesidad de ejecutarlo. Esto hace posible que el buscador se pueda desplegar independientemente del resto de etapas en cualquier momento, siempre y cuando se haya mantenido la configuración general del índice utilizada.

De forma opcional y únicamente si se desea aplicar *reranking*, se necesitará acceso al subconjunto de métricas, que ya ha sido generado *offline*.

4.4.1. Buscador de entidades

El buscador de entidades se encarga de realizar consultas sobre el sistema de recuperación de información final, que en nuestro caso es *ElasticSearch*.

Además de servir como conector, también posee la capacidad de hacer *reranking*, es decir, de reordenar los resultados obtenidos del sistema con criterios propios. Para ello, se realiza una media geométrica ponderada de la misma forma que para los agregadores de métricas, donde en este caso se aplica a nivel de entidad entre la puntuación devuelta por el sistema de recuperación de información y cualquier otro criterio. Esto se puede ver en la Fórmula 2.

$$score_{reranking}(r,\ peso) = score_{criterio}(r)^{peso} \cdot score_{sistema}(r)^{(1-peso)} \tag{2}$$

En este caso, el criterio propio aplicado es el *InfoRank* de las entidades devueltas. Aplicado a un número limitado de resultados, como por ejemplo los primeros 10, se puede conseguir reordenarlos según sus importancias, que vienen dictadas por su informatividad y relaciones con el resto de entidades, gracias a dicha métrica.

Esto puede resultar de gran ayuda en casos donde la consulta original no ha conseguido devolver ninguno o pocos resultados aceptables, en cuyo caso un criterio alternativo de ordenación es preferible a las puntuaciones ofrecidas por el sistema de recuperación, que no serán interpretables al no haber buenos resultados. En estas situaciones, se puede optar por al menos visualizar las entidades más relevantes, según *InfoRank*, primero.

Adicionalmente, se puede utilizar para reordenar los primeros resultados devueltos en casos en los que el usuario observe que se ha devuelto una entidad relacionada pero irrelevante al objetivo de la consulta. Un ejemplo de este último caso se puede encontrar en la Figura 10.

4.4.2. Interfaz de búsqueda

Con el objetivo de mostrar el funcionamiento del sistema de cara al usuario, hemos desarrollado una interfaz gráfica de búsqueda que integra tanto el componente de búsqueda sobre el sistema de recuperación de información final como la funcionalidad de *reranking*.

De esta forma, es posible buscar y realizar *reranking* dentro de la misma consulta si se observan posibles resultados espurios, así como acceder a cada uno de los recursos de la *DBpedia* desde el navegador, al ser todas las entidades *URIs* válidas. Un ejemplo de la misma se encuentra en la Figura 11.

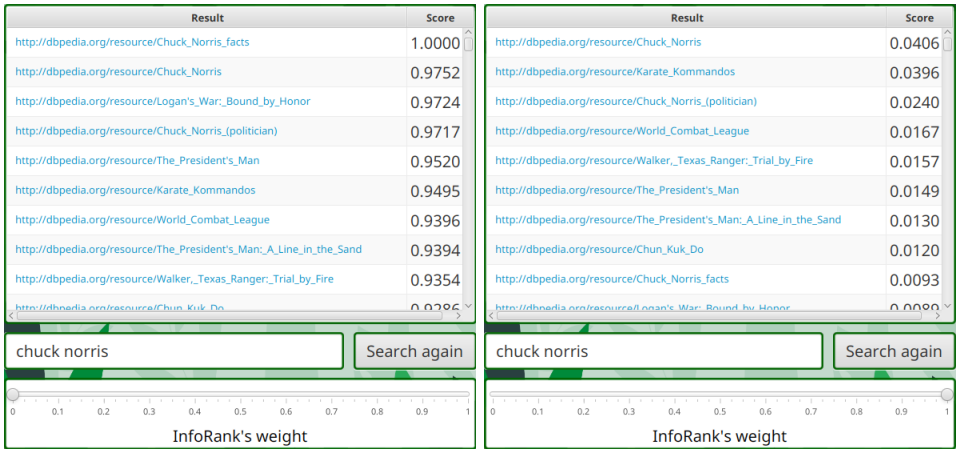

Figura 10: Ejemplo de *reranking* mediante *InfoRank*. Al buscar en el sistema "*chuck norris*", el modelo utilizado por el sistema pesaba incorrectamente más el artículo sobre los *facts* de Chuck Norris que su propia entidad. Tras reordenar los 10 primeros resultados mediante *InfoRank* exclusivamente, se devuelven su entidad y sus apariciones en películas primero, al contar con más relaciones e información.

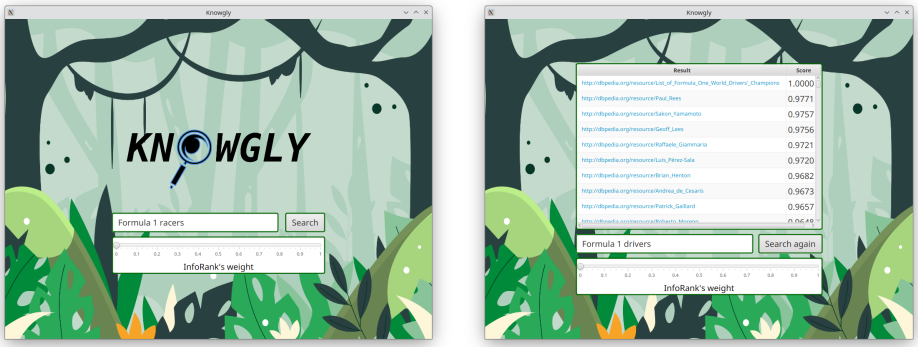

Figura 11: Interfaz de búsqueda de cara al usuario. La funcionalidad de *reranking* se expone mediante un selector que controla el peso a dar a la puntuación de *InfoRank* de cada entidad. De esta forma, el usuario puede reordenar resultados en casos en los que observe que alguno pueda no ser relevante a su consulta.

4.5. Etapa de evaluación

La etapa de evaluación se puede considerar como una etapa completamente separada del resto, implementada exclusivamente para comprobar el comportamiento del sistema frente a diferentes configuraciones internas y clusterizaciones de agregadores de métricas.

4.5.1. Generador de resultados

El generador de resultados tiene como función realizar una serie de búsquedas sobre el sistema de recuperación de información final, recopilando los resultados obtenidos para cada una de ellas. Tanto las consultas, introducidas por fichero, como los resultados a escribir se encuentran en el formato *TREC* estándar, implementado por la herramienta de evaluación *trec_eval*[23].

Este formato y su herramienta son utilizadas para evaluar resultados de sistemas de recuperación de información de cualquier tipo y para un gran número de tareas, entre las que se encuentra la búsqueda orientada a entidades. Esto permite al sistema delegar la tarea de evaluación a un programa especializado en ello, capaz de generar un gran número de métricas configurables a partir de un mismo fichero de resultados.

En este caso, las consultas de entrada son las pertenecientes al conjunto de evaluación de *DBpedia-Entity*, descrito en la Sección 3.4, y las salidas son alimentadas a *trec_eval*, obteniendo así las medidas $NDCG@10$ y $NDCG@100$. Las búsquedas se realizan y repiten con diferentes estrategias, que en el caso de *ElasticSearch* implican utilizar diferentes modelos de consultas, que se detallan en la Sección 5. Así mismo, se repiten con diferentes ponderaciones de *reranking* con la medida de *InfoRank*.

4.5.2. Evaluador del sistema

El evaluador del sistema se encarga de generar diferentes esquemas de indexación. Para ello, parte de una lista de todos los agregadores de métricas definidos, de tal forma que genera un esquema de indexación a partir de cada uno de ellos y ejecuta toda la etapa de extracción e indexación. Una vez realizado, ejecuta el generador de resultados, que prueba diferentes estrategias de búsqueda sobre el índice generado y almacena los resultados.

Esto se realiza también para todas las combinaciones posibles entre agregadores de métricas de ambas familias de métricas, probando diferentes ponderaciones entre ellos.

Este módulo ha sido utilizado para realizar un análisis exhaustivo y automático del comportamiento del sistema, obteniendo la mejor configuración general y combinación de agregadores de métricas para el conjunto de pruebas, que posteriormente se ha analizado de forma manual. Este proceso se detalla en la Sección 5.

[23] https://github.com/usnistgov/trec_eval

5. Resultados

Una vez establecidas todas las configuraciones expuestas por *Knowgly*, hemos utilizado el conjunto de pruebas de *DBpedia-Entity* para establecer las mejores configuraciones y realizar una comparativa con los sistemas existentes. Para ello, hemos realizado una serie de pruebas automáticas y manuales generando el mismo número de resultados, que en este caso son 1000 entidades a devolver por consulta, y los hemos evaluado con *trec_eval*, solicitando las medidas $NDCG@10$ y $NDCG@100$.

Por otro lado, hemos establecido una comparativa en función de las ventajas y desventajas que posee la aproximación seguida por *Knowgly*.

5.1. Comparativa cualitativa frente a los sistemas actuales

Debido a que la metodología utilizada por *Knowgly* es considerablemente diferente a la utilizada por los sistemas tradicionales, primero se debe realizar una comparativa cualitativa, con el objetivo de establecer las ventajas y desventajas de *Knowgly* frente a otros sistemas. Siguiendo el estado del arte actual, se pueden distinguir dos tipos de sistemas de búsqueda de entidades: los basados en índices realizados manualmente, a los que o bien se les aplica modelos de recuperación de información clásicos o basados en aprendizaje supervisado, y los basados en modelos de lenguaje modernos.

5.1.1. Generación de índices

Los sistemas basados en la conversión de entidades del grafo de conocimiento a documentos virtuales, almacenados en índices de sistemas como *ElasticSearch*, realizan las mismas tareas que *Knowgly*. La diferencia radica en su método de generación del índice, ya que este esquema, y por tanto la asignación de predicados a sus campos, se realiza manualmente. Esto requiere un análisis exhaustivo manual o semi-automático del grafo de conocimiento, que es inabordable a medida que el tamaño de los mismos crece.

Tomando como ejemplo la versión utilizada del grafo de la *DBpedia*, podemos encontrar en ella 61.828 predicados, cuya asignación a las entidades varía enormemente en función de los 385 tipos posibles que pueden tener asignados. Esto hace difícil establecer la importancia de predicados no globales, es decir, de aquellos que solo se encuentran relacionados con entidades de cierto tipo.

Del mismo modo, se pueden encontrar predicados distintos que cumplen la misma función en diferentes niveles de la jerarquía de tipos, como $<$*dbp:divisionName*$>$ y $<$*dbp:subdivisionName*$>$. Al hacer una simple búsqueda de predicados que acaben en $*$*Name*, es posible encontrar 558 de ellos. Realizar un análisis manual de cada uno de esos predicados, así como del resto de predicados para cada uno de los significados semánticos posibles que pueden asumir en el grafo, requiere de un gran esfuerzo y de un juicio de relevancia empírico. Esto se traduce en la posibilidad de incluir por error predicados espurios en un campo prioritario del índice, o incluso de omitir predicados válidos.

Todo esto obliga a generar un índice en alto nivel, eligiendo solamente unos pocos predicados generales, considerados como relevantes siguiendo algún criterio empírico. Por ejemplo, en el índice utilizado por los sistemas de *DBpedia-Entity*,

que se encuentra en el Anexo A, solo se han elegido 21 de los más de 558 posibles predicados de nombre, un predicado que indica las categorías de la *Wikipedia* y tres predicados para las redirecciones, desambiguaciones y enlaces entre artículos, agrupando el resto en campos genéricos.

5.1.2. Modelos de recuperación clásicos

Los modelos de recuperación clásicos incluyen modelos comúnmente utilizados desde hace décadas, como *BM25* [15], *LM* [22], *PRMS* [9] o *SDM* [11].

Todos ellos realizan diferentes suposiciones sobre la distribución de los términos en los documentos, y se basan en la utilización de un único campo. Este campo, en el contexto de la búsqueda orientada a entidades, es justamente el *catchAll*, el cual contiene la representación textual de todas las relaciones y atributos de una entidad.

Estos modelos no son óptimos para la búsqueda de entidades, al poder generar fácilmente falsos positivos. Esto se debe a que una entidad puede contener un número elevado de relaciones con otras, que se traduce en menciones textuales a ellas en el *catchAll*. Esto provoca casos en los que, al buscar menciones de una entidad r, otra entidad s que tenga muchas relaciones con r tenga más peso que la propia entidad r. Al haber introducido todo el contenido textual en un único campo, se pierde la posibilidad de distinguir o pesar la relevancia de las relaciones, y por tanto de corregir estos problemas.

5.1.3. Modelos de recuperación con aprendizaje supervisado

Los modelos de recuperación basados en la utilización de técnicas como *Learning to Rank* utilizan comúnmente modelos de recuperación basados en campos, como *BM25F* [15], *MLM* [12] o *FSDM* [23]. Todos estos modelos se basan en la idea de aplicar múltiples instancias de algoritmos clásicos sobre los campos de un documento, pesándolos con prioridades diferentes y combinando las puntuaciones finales para todos ellos.

Debido a que normalmente cuentan con una instancia configurable individual en cada campo, es posible ajustar sus parámetros internos por separado, además de poder ajustar los pesos asignados a cada campo, de tal forma que unos tengan más prioridad que otros. Esto provoca que, aunque el modelo final pueda ser adaptado al esquema con mucha precisión, el número de parámetros a ajustar pueda ser excesivo, debiendo recurrir a técnicas como el *Learning to Rank*.

Aunque estos métodos cuentan con la ventaja de ser los que mejores resultados ofrecen, aplicar aprendizaje supervisado conlleva la necesidad de poseer un conjunto de entrenamiento y de validación, que en este caso son consultas y posibles resultados. Como ya se ha mencionado en la Sección 1.2, esto es una tarea difícil de realizar en el ámbito de la evaluación de sistemas de recuperación de información, ya que es costoso en recursos, y los datos utilizados para entrenar y evaluar pueden a su vez no ser exhaustivos o representativos del uso real del sistema.

En comparación con estos modelos, nuestro sistema no realiza ningún ajuste automático de los parámetros de los modelos de recuperación que utilizamos, que son *BM25F*, *BM25* y *LM*, y los pesos de los campos son estáticos, al ser marcados por la configuración. Por este motivo, es difícil rivalizar los resultados que ofrecen.

En cambio, *Knowgly* ofrece la ventaja de mitigar la necesidad de optimizar el modelo para un índice, dado que el esquema de distribución de predicados ya se encuentra optimizado en función de la estructura del grafo y las métricas utilizadas. Esto a su vez elimina el requisito de poseer un conjunto de entrenamiento, ya que todo este proceso se basa el algoritmo de *KMeans*, que es una técnica de *Machine Learning* no supervisada.

5.1.4. Sistemas basados en modelos de lenguaje

Los sistemas basados en lenguaje se fundamentan en el uso de los *LLM* (*Large Language Model*) como *BERT* [2] o *GPT* [14]. En el campo de la búsqueda orientada a entidades, estos modelos se basan actualmente en la generación de representaciones densas de entidades [13] y en el re-entrenamiento de dichos modelos para la tarea objetivo, en función de un conjunto de datos como el de *DBpedia-Entity*. De esta forma, realizan un *reranking* óptimo de los resultados obtenidos por modelos de recuperación como *BM25* o *BM25F*.

Actualmente, estos modelos son los que mejores resultados ofrecen, alcanzando una puntuación de 0.541 [3] en la medida $NDCG@10$ para *DBpedia-Entity* frente al mejor resultado obtenido previamente de 0.4605 para *BM25F-CA*. Todos estos resultados se encuentran en el Anexo A.

Del mismo modo que en el caso anterior, *Knowgly* no aspira, de momento, a rivalizar con los resultados de estas redes neuronales. Sin embargo, sigue contando con la misma ventaja de no requerir ningún tipo de entrenamiento, que en el caso de dichos modelos es mucho más costoso. Esto se debe a que, además de requerir de un conjunto de evaluación tradicional, estos sistemas requieren de los siguientes elementos:

- Un sistema que proporcione representaciones densas, denominadas *embeddings*[24], de elementos de las tripletas de los grafos de conocimiento, como *Wikipedia2Vec* [21], que a su vez debe ser entrenado con fuentes de datos textuales y grafos de conocimiento existentes.
- Un componente que identifique entidades en las consultas y los documentos sobre los que entrenar el sistema, mediante *NER* (*Named Entity Recognition*[25]). Para ello, se utilizan sistemas como *REL* [8], que a su vez se basan en el uso de representaciones densas de *Wikipedia2Vec*.

Esto provoca que dichos sistemas se adapten exclusivamente a una única fuente de datos y a consultas similares a las usadas en el entrenamiento y evaluación. Por consecuencia, requieren adaptaciones y re-entrenamientos, ya de por sí costosos, para adaptarlos a fuentes de datos diferentes, como grafos de conocimiento de un ámbito reducido, o a consultas de diferente naturaleza, como las de *Question Answering*[26].

[24] https://en.wikipedia.org/wiki/Embedding
[25] https://en.wikipedia.org/wiki/Named-entity_recognition
[26] https://en.wikipedia.org/wiki/Question_answering

5.2. Comparativa cuantitativa frente a los sistemas actuales

A continuación se muestran los resultados obtenidos por *Knowgly* en el conjunto de pruebas de *DBpedia-Entity*, así como las conclusiones extraídas de su análisis.

5.2.1. Método de evaluación

Para conseguir los resultados del sistema hemos realizado un análisis exhaustivo de todas las configuraciones que exponemos, buscando aquellas con los mejores resultados para las métricas $NDCG@10$ y $NDCG@100$, que son las utilizadas en *DBpedia-Entity*. Este análisis se ha realizado con un primer paso automatizado, debido al elevado número de configuraciones a evaluar, y un análisis y refinamiento manuales una vez encontradas las mejores configuraciones.

El proceso de evaluación automático se ha centrado en el análisis de los esquemas de índice generados por cada agregador de métricas, que son los que van a producir mayores variaciones en los resultados. Para ello, hemos generado y evaluado índices de forma automática tanto con los agregadores individuales como con las combinaciones entre los de importancia estructural e *InfoRank*, con ponderaciones de 0.25, 0.50 y 0.75.

Este análisis ha partido de una base de configuración común para el resto de elementos del sistema, con el objetivo de poder asociar los cambios de puntuaciones a los agregadores de métricas exclusivamente, que consiste en lo siguiente:

- **Diferentes representaciones textuales de documentos**: Hemos decidido utilizar una representación textual básica en todo momento para partir de una base común, que hemos enriquecido durante las pruebas manuales tal y como se describe en el Anexo A.1.1.

- **Técnicas de tratamiento de la entrada**: Hemos aplicado una serie de tratamientos de la entrada, que consisten en el paso del texto a minúsculas, la eliminación de palabras vacías y el *Stemming*, que se encuentran detallados en el Anexo A.1.2.

- **Modelos de recuperación y consulta**: En nuestro caso, hemos utilizado los modelos de recuperación *BM25* y *BM25F* [15], cuyos parámetros no se han ajustado en ningún momento, utilizando en su lugar sus valores por defecto. Durante las pruebas manuales posteriores hemos analizado también el modelo de *LM* [22] y ajustado los parámetros de todos ellos, que se detallan en el Anexo A.1.3.

  Por otro lado, hemos utilizado los modelos de consulta de *ElasticSearch*, aprovechando su implementación de *BM25F* y definiendo una consulta booleana propia denominada *BoolQuery*. Todo esto se detalla en el Anexo A.1.4.

- **Conversión de entidades a contenido textual**: La conversión a texto e indexación de las entidades del grafo se ha realizado siguiendo los criterios de los sistemas evaluados originalmente en *DBpedia-Entity*, para poder así evaluar el sistema con una colección idéntica de documentos. Esto se encuentra detallado en el Anexo A.1.5.

- **Configuraciones de agregadores de métricas**: Hemos analizado un gran número de configuraciones para los analizadores de métricas, como el número

de campos a utilizar, los pesos asignados a cada uno de ellos o la separación de predicados entre *Object Properties* y *Datatype Properties* durante la clusterización. Estos aspectos se encuentran detallados en los Anexos A.1.6, en el cual se detallan estas configuraciones, y A.1.7, en el que se detalla en profundidad el razonamiento detrás de las diferentes estrategias de pesos utilizadas.

Durante las pruebas automáticas, hemos analizado todos los agregadores de métricas y sus posibles combinaciones con las siguientes configuraciones:

- 3 clústeres, con pesos [1.0, 0.1, 0.05].
- 3 clústeres, separando *Object Properties* y *Datatype Properties* (obteniendo así 6 clústeres) con pesos [1, 0.05, 0.05] para *Datatype Properties* y [0.2, 0.05, 0.0] para *Object Properties*.
- 5 clústeres, con pesos [1.0, 0.5, 0.15, 0.10, 0.05].

En el Anexo B se pueden encontrar los resultados de todas las pruebas automáticas.

5.2.2. Resultados de Knowgly

A partir del análisis en profundidad realizado, hemos elegido el agregador de métricas (o combinación de ellos) con la mejor puntuación en cada una de las tres configuraciones. A partir de sus configuraciones de índice, que hemos enriquecido con una representación adicional de *keywords* y analizado con *LM* y *BM25* con parámetros ajustados, hemos realizado una ejecución exhaustiva de las consultas con diferentes variaciones de pesos, obteniendo así los resultados finales, que se pueden encontrar en las Tablas 1, 2 y 3.

InfoRank de predicados														
Modelo de consulta	Pesos			SemSearch ES		INEX-LD		ListSearch		QALD-2		Total	
	Campo 1	Campo 2	Campo 3	@10	@100	@10	@100	@10	@100	@10	@100	@10	@100
BoolQuery (BM25)	1.0	0.05	0.0	0.6144	**0.6729**	0.3958	0.4406	**0.3996**	**0.4500**	0.3270	0.3972	**0.4290**	**0.4861**
BoolQuery (LM)	1.0	0.0	0.05	**0.6174**	0.6697	0.3903	**0.4472**	0.3902	0.4419	**0.3286**	**0.4009**	0.4267	0.4858
BM25F (Text)	20.0 (1.0)	5.0 (0.25)	2.0 (0.1)	0.5928	0.6620	**0.3972**	0.4375	0.3760	0.4278	0.3082	0.3768	0.4126	0.4712
CatchAll (BM25)	-	-	-	0.5538	0.6373	0.3625	0.4139	0.3756	0.4277	0.3159	0.3808	0.3980	0.4614
CatchAll (LM)	-	-	-	0.5526	0.6355	0.3522	0.4174	0.3675	0.4262	0.3157	0.3842	0.3935	0.4624
BoolQuery en CatchAll (BM25)	-	-	-	0.5726	0.6365	0.3667	0.4169	0.3746	0.4274	0.3159	0.3808	0.4032	0.4618
BoolQuery en CatchAll (LM)	-	-	-	0.5709	0.6334	0.3582	0.4197	0.3678	0.4268	0.3157	0.3842	0.3993	0.4625

Tabla 1: Resultados para la clusterización en 3 campos con *Predicate-based InfoRank*.

InfoRank de predicados																
Modelo de consulta	Pesos					SemSearch ES		INEX-LD		ListSearch		QALD-2		Total	
	Campo 1	Campo 2	Campo 3	Campo 2	Campo 3	@10	@100	@10	@100	@10	@100	@10	@100	@10	@100
BoolQuery (BM25)	1.0	0.5	0.05	0.05	0.05	0.6344	0.6774	0.3736	0.4069	0.3448	0.3858	0.2802	0.3358	0.4016	0.4458
BoolQuery (LM)	1.0	1.0	0.05	0.05	0.00	**0.6458**	**0.6997**	**0.4026**	**0.4487**	**0.3918**	**0.4338**	**0.3380**	**0.4024**	**0.4394**	**0.4919**
BM25F (Text)	20.0 (1.0)	4.0 (0.2)	1.0 (0.05)	1.0 (0.05)	1.0 (0.05)	0.5662	0.6412	0.3293	0.3869	0.3540	0.4132	0.2788	0.3335	0.3776	0.4389
CatchAll (BM25)	-	-	-	-	-	0.5579	0.6403	0.3613	0.4141	0.3758	0.4260	0.3127	0.3793	0.3979	0.4613
CatchAll (LM)	-	-	-	-	-	0.5548	0.6367	0.3557	0.4188	0.3667	0.4222	0.3135	0.3828	0.3939	0.4615
BoolQuery en CatchAll (BM25)	-	-	-	-	-	0.5784	0.6397	0.3652	0.4167	0.3741	0.4260	0.3127	0.3793	0.4032	0.4617
BoolQuery en CatchAll (LM)	-	-	-		-	0.5710	0.6332	0.3615	0.4210	0.3669	0.4230	0.3135	0.3828	0.3991	0.4614

Tabla 2: Resultados para la clusterización en 5 campos con *Predicate-based InfoRank*.

InfoRank de predicados															
Modelo de consulta	Tipo de predicados	Pesos			SemSearch ES		INEX-LD		ListSearch		QALD-2		Total	
		Campo 1	Campo 2	Campo 3	@10	@100	@10	@100	@10	@100	@10	@100	@10	@100
BoolQuery (BM25)	**Datatype**	1.0	0.05	0.05	**0.5743**	**0.6460**	0.4061	0.4421	0.3907	0.4297	0.3334	0.3954	0.4212	0.4744
	Object	0.2	0.05	0.0										
BoolQuery (LM)	**Datatype**	1.0	0.05	0.05	0.5710	0.6364	**0.4082**	**0.4548**	**0.3923**	**0.4392**	**0.3407**	**0.4100**	**0.4234**	**0.4815**
	Object	0.4	0.1	0.0										
BM25F (Text)	**Datatype**	20.0 (1.0)	2.0 (0.1)	1.0 (0.05)	0.5662	0.6348	0.3165	0.3678	0.3502	0.4060	0.2521	0.3070	0.3659	0.4236
	Object	20.0 (1.0)	2.0 (0.1)	1.0 (0.05)										
CatchAll (BM25)	**Datatype**	-	-	-	0.5530	0.6373	0.3629	0.4146	0.3762	0.4283	0.3166	0.3811	0.3983	0.4618
	Object	-	-	-										
CatchAll (LM)	**Datatype**	-	-	-	0.5527	0.6357	0.3520	0.4173	0.3674	0.4257	0.3160	0.3845	0.3936	0.4624
	Object	-	-	-										
BoolQuery en CatchAll (BM25)	**Datatype**	-	-	-	0.5721	0.6364	0.3670	0.4176	0.3753	0.4281	0.3166	0.3811	0.4036	0.4622
	Object	-	-	-										
BoolQuery en CatchAll (LM)	**Object**	-	-	-	0.5710	0.6333	0.3580	0.4198	0.3677	0.4263	0.3160	0.3845	0.3993	0.4625
	Object	-	-	-										

Tabla 3: Resultados para la clusterización en 3 campos con *Predicate-based InfoRank*, separando *Datatype properties* y *Object properties*.

Un aspecto a tener en cuenta son las variaciones de las puntuaciones obtenidas para el campo *CatchAll*, que se deben a ligeros cambios en las longitudes de los documentos al introducir el mismo contenido textual en diferente orden. En este caso, hemos optado por tomar como referencia para *CatchAll* su mejor puntuación obtenida entre todos los casos, que es 0.4036 tras los ajutes anteriormence mencionados.

En el caso de nuestros esquemas, la mejor puntuación obtenida ha sido de 0.4394, con una división en 5 clústeres.

5.2.3. Análisis de los resultados

Como se puede observar en las Tablas 1, 2 y 3, hemos conseguido superar las puntuaciones obtenidas al realizar las consultas sobre el campo *catchAll*, con y sin representaciones textuales adicionales, que es el equivalente a la aproximación básica, sin aprovechar la división en campos creada por *Knowgly*.

Por otro lado, comparando los mejores resultados con los de *DBpedia-Entity*, que se pueden encontrar en la Tabla 5 del Anexo A, hemos superado los obtenidos

por todos los modelos de recuperación básicos, como *BM25*, *PRMS*, *LM* y *SDM*. Adicionalmente, hemos conseguido rivalizar con los modelos de recuperación optimizados mediante aprendizaje supervisado, como *MLM-CA* y *BM25-CA*. En el caso de *BM25F*, las limitaciones de *ElasticSearch* a la hora de asignar pesos a sus campos, que no permiten aplicar penalizaciones a los pesos o utilizar diferentes representaciones textuales, no nos han permitido optimizarlo tanto como deseábamos.

Estos resultados han sido muy satisfactorios, especialmente teniendo en cuenta que el ajuste de los pesos y modelos de recuperación los hemos realizado manualmente sin ningún tipo de *Machine Learning* supervisado, algo que se requería hacer en los sistemas de *DBpedia-Entity*. Esto nos lleva a la conclusión de que la premisa de *Knowgly*, basada en la generación de un esquema de índice óptimo para evitar tener que aplicar técnicas de aprendizaje supervisado, ha resultado ser efectiva.

En el Anexo C se puede encontrar un análisis en profundidad de otros aspectos del sistema, como la significación estadística de los resultados, su comportamiento frente a cada tipo de consulta de *DBpedia-Entity*, el efecto producido por las diferentes estrategias de asignación de pesos a los campos, las diferencias observadas entre los generadores de métricas o el comportamiento de *BM25* frente a *LM*.

6. Conclusiones

La realización de este trabajo me ha permitido reforzar bases teóricas previas y profundizar mucho más en áreas como la recuperación de información, las tecnologías de la Web Semántica o el procesamiento del lenguaje natural. Así mismo, como consecuencia del aspecto innovativo del trabajo, he podido realizar un estudio en profundidad del estado del arte relativo a la recuperación de información semántica. Esto ha supuesto una labor previa de análisis de un gran número de publicaciones científicas, ya sea para aprender sobre el contexto de las tareas realizadas como para poder obtener una visión crítica del estado del arte, algo a lo que no me había enfrentado anteriormente.

Durante la implementación del sistema, he podido profundizar en el diseño de arquitecturas software de gran envergadura, debiendo tomar decisiones de diseño sobre las clases e interfaces para acomodar cambios futuros, facilitar su comprensión y permitir su inclusión en contextos diferentes a la búsqueda orientada a entidades.

Así mismo, me he podido enfrentar a tareas complejas como la optimización de sistemas para trabajar con conjuntos de datos de elevado tamaño, bajando de nivel de abstracción al operar con los mismos en situaciones con requisitos de tiempo y espacio exigentes. Esto lo he tenido que realizar en muchos momentos, como durante la aplicación de métricas sobre los grafos de conocimiento o en la indexación, donde implementaciones de algoritmos ingenuas son inadmisibles, tanto en tiempo como en espacio, con grandes volúmenes de datos.

Por otro lado, he podido trabajar con la paralelización masiva de algoritmos secuenciales, analizando las mejores formas de aplicar algoritmos para minimizar la contención entre los hilos, en situaciones donde cualquier sincronización innecesaria supone grandes penalizaciones en tiempo, como durante la aplicación métricas.

Por último, el uso de conjuntos de evaluación me ha permitido realizar un análisis crítico del sistema, evaluando una gran multitud de configuraciones y comportamientos para obtener los mejores resultados posibles, tanto automática como manualmente. En el proceso, he podido profundizar en las técnicas que se utilizan para evaluar sistemas de recuperación de información en publicaciones científicas, como los formatos estándar, las métricas de evaluación o la metodología para documentar experimentos.

Este trabajo se ha desarrollado bajo el contexto de una beca de colaboración con el grupo de investigación SID[27], en la propia universidad de Zaragoza, donde *Knowgly* ha formado parte de los sistemas desarrollados en ella. Debido a esto, el sistema desarrollado en este trabajo va a seguir recibiendo nuevas funcionalidades, con el objetivo de mejorar sus resultados.

Como objetivo de cara a futuro, esperamos publicar como un artículo de investigación todos los aspectos sobre *Knowgly*, tal y como se han detallado en este documento, incluyendo las métricas utilizadas y los resultados de futuras extensiones.

[27] http://sid.cps.unizar.es/

6.1. Cronograma

En la Figura 12 se muestra la división de tareas y el cómputo final de horas de dedicación. Debido a su realización dentro del contexto de una beca de colaboración, el trabajo ha constado de una parte considerable de estudio previo de bibliografía y artículos de investigación, relacionados con la recuperación de información semántica.

Tarea	Horas
Estudiar bibliografía y artículos de investigación	47
Revisar documentación de librerías a utilizar	19
Estudiar y preparar el conjunto de pruebas a utilizar	24
Implementar el extractor de entidades	12
Implementar el indexador de entidades	17
Implementar el buscador de entidades	9
Implementar métricas de importancia estructural	13
Implementar métricas de InfoRank	25
Implementar PageRank bidireccional	8
Implementar agregadores de métricas	15
Implementar pipeline de etapas automatizado	18
Reimplementar métricas en HDT y paralelizar algoritmos	62
Reimplementar indexador de entidades en HDT	4
Implementar pipeline de evaluación automatizado	8
Documentar el proyecto y escribir la memoria	66
Evaluar configuraciones del sistema y de ElasticSearch	93
Total:	440

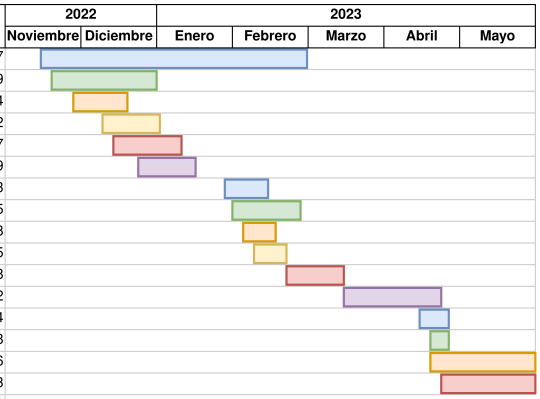

Figura 12: División de tareas y cómputo final de horas de dedicación.

En general, el tiempo dedicado a la implementación de todos los módulos ha sido equitativa, con la excepción de la reimplementación de los generadores de métricas en *HDT*. Esto último ha requerido la realización de múltiples pruebas de carga y análisis exhaustivos de las optimizaciones realizadas, para verificar que las métricas se habían replicado correctamente y a su vez conseguir que el sistema sea adaptable a grafos de conocimiento de gran tamaño.

Por otro lado, la evaluación del comportamiento del sistema ha sido la tarea que ha requerido la mayor dedicación. Esto es debido a que, además de realizar evaluaciones automáticas, se ha llevado a cabo un gran número de análisis manuales de resultados para verificar hipótesis, como distintos agregadores de métricas o métodos de indexación, y comprobar el efecto de nuevas configuraciones del sistema y de *ElasticSearch*.

6.2. Trabajo futuro

A continuación se describen las extensiones planteadas actualmente, que se esperan implementar como parte de la beca de colaboración.

6.2.1. Generación del esquema de indexación basada en tipos

Una alternativa a distribuir predicados a lo largo de los campos del índice en función de su importancia global es realizarla localmente, a nivel de cada entidad. Aunque con las métricas actuales solo se puede inferir a nivel de entidad la importancia de las mismas, es posible obtener importancias de predicados desambiguadas por tipo. Basándose en esto, es posible generar un esquema de distribución de predicados en función de únicamente los tipos de una entidad.

Esta aproximación permite un nivel de granuralidad mucho más fino, ya que cada entidad tendrá una organización de sus predicados a lo largo de los campos localmente óptima y diferente al del resto de entidades, manteniendo la transparencia de cara a las búsquedas.

Un ejemplo de este comportamiento en la *DBpedia* sería la diferente distribución de predicados para el tipo <*dbo:Place*>, frente a un subtipo más específico en la jerarquía como <*dbo:NaturalPlace*>. En este caso, el peso para el predicado <*dbp:riverName*> es mucho mayor para <*dbo:NaturalPlace*>, al aparecer con mucha más frecuencia en instancias con ese tipo. Esto consigue otorgar más importancia a predicados asociados comúnmente a entidades de un tipo dado.

Una primera aproximación se encuentra ya desarrollada en el sistema, quedando pendiente un estudio detallado del comportamiento de posibles estrategias a seguir en el caso de que una entidad tenga más de un tipo, debiendo en estos casos generar una fusión de varias distribuciones de predicados distintas.

6.2.2. Tratamiento de la entrada

El sistema desarrollado actualmente se fundamenta en el uso de consultas basadas en palabras clave, como "*Formula 1 racers*" o "*famous british actors*". Uno de los objetivos de cara a futuro es permitir la inclusión de consultas formuladas en lenguaje natural, como "Quiero una lista de pilotos de Formula 1" o "Actores Irlandeses o Escoceses famosos".

Esto requiere de un módulo de tratamiento de la entrada, de tal forma que sea posible distinguir entre ambos tipos de consultas, y tras ello tratarlas de forma separada con herramientas de procesamiento de lenguaje natural como el *Entity Linking*, infiriendo qué porciones de la consulta se refieren a una entidad dada. Esto permitiría dividir y procesar una consulta como subconsultas booleanas.

6.2.3. Paso de mensajes

Del mismo modo que el sistema no está preparado actualmente para recibir consultas en lenguaje natural, tampoco es capaz de afrontar consultas de *Question Answering*, como pueden ser "Año de nacimiento de David Bowie" o "Lenguajes hablados en Suiza".

Para conseguir afrontar este tipo de consultas, se ha planteado realizar una implementación de la estrategia de paso de mensajes [16], que utiliza un método no supervisado para obtener posibles objetos que respondan a la pregunta planteada, partiendo de un conjunto de entidades asociadas a dicha consulta mediante *Entity Linking*. Al mismo tiempo, se ha planteado investigar las técnicas utilizadas en el paso de mensajes, con el objetivo de analizar semejanzas con las utilizadas en *Knowgly*, y así encontrar posibles mejores en cualquiera de los dos.

6.2.4. Uso de otros grafos de conocimiento

El uso del grafo de la *DBpedia* nos ha permitido evaluar el comportamiento de *Knowgly* ante grafos de propósito general de gran tamaño, con un gran número de datos espurios.

De cara a futuro, queremos evaluar su comportamiento con grafos de tamaño más limitado y un dominio específico, como son los grafos de *MusicBrainz*[28], un repositorio de datos de música colaborativo, e *IMDb*[29], un repositorio similar relacionado con la cinematografía. Para ello, pensamos utilizar los conjuntos de evaluación desarrollados en *QUIRA* [10], que se utilizaron originalmente para evaluar los resultados de *InfoRank* sobre ambos grafos.

[28] https://musicbrainz.org/
[29] https://www.imdb.com/

7. Bibliografía

[1] Krisztian Balog. *Entity-Oriented Search*. 1st. Springer Publishing Company, Incorporated, 2018. ISBN: 3319939335.

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[3] Emma J. Gerritse, Faegheh Hasibi y Arjen P. de Vries. "Entity-aware Transformers for Entity Search". En: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul. de 2022. DOI: 10.1145/3477495.3531971. URL: https://doi.org/10.1145%2F3477495.3531971.

[4] Thomas R. Gruber. "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". En: *Int. J. Hum.-Comput. Stud.* 43.5–6 (dic. de 1995), págs. 907-928. ISSN: 1071-5819. DOI: 10.1006/ijhc.1995.1081. URL: https://doi.org/10.1006/ijhc.1995.1081.

[5] Faegheh Hasibi, Krisztian Balog y Svein Erik Bratsberg. "Dynamic Factual Summaries for Entity Cards". En: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, págs. 773-782. ISBN: 9781450350228. DOI: 10.1145/3077136.3080810. URL: https://doi.org/10.1145/3077136.3080810.

[6] Faegheh Hasibi, Krisztian Balog y Svein Erik Bratsberg. "Exploiting Entity Linking in Queries for Entity Retrieval". En: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*. ICTIR '16. Newark, Delaware, USA: Association for Computing Machinery, 2016, págs. 209-218. ISBN: 9781450344975. DOI: 10.1145/2970398.2970406. URL: https://doi.org/10.1145/2970398.2970406.

[7] Faegheh Hasibi et al. "DBpedia-Entity v2: A Test Collection for Entity Search". En: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, págs. 1265-1268. ISBN: 9781450350228. DOI: 10.1145/3077136.3080751. URL: https://doi.org/10.1145/3077136.3080751.

[8] Johannes M. van Hulst et al. "REL: An Entity Linker Standing on the Shoulders of Giants". En: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul. de 2020. DOI: 10.1145/3397271.3401416. URL: https://doi.org/10.1145%2F3397271.3401416.

[9] Jinyoung Kim, Xiaobing Xue y W. Bruce Croft. "A Probabilistic Retrieval Model for Semistructured Data". En: *Advances in Information Retrieval*. Ed. por Mohand Boughanem et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, págs. 228-239. ISBN: 978-3-642-00958-7.

[10] Elisa Menendez et al. "Novel Node Importance Measures to Improve Keyword Search over RDF Graphs". En: ago. de 2019, págs. 143-158. ISBN: 978-3-030-27617-1. DOI: 10.1007/978-3-030-27618-8_11.

[11] Donald Metzler y W. Bruce Croft. “A Markov Random Field Model for Term Dependencies”. En: *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’05. Salvador, Brazil: Association for Computing Machinery, 2005, págs. 472-479. ISBN: 1595930345. DOI: 10.1145/1076034.1076115. URL: https://doi.org/10.1145/1076034.1076115.

[12] Paul Ogilvie y Jamie Callan. “Combining Document Representations for Known-Item Search”. En: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*. SIGIR ’03. Toronto, Canada: Association for Computing Machinery, 2003, págs. 143-150. ISBN: 1581136463. DOI: 10.1145/860435.860463. URL: https://doi.org/10.1145/860435.860463.

[13] Nina Poerner, Ulli Waltinger e Hinrich Schütze. “E-BERT: Efficient-Yet-Effective Entity Embeddings for BERT”. En: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, nov. de 2020, págs. 803-818. DOI: 10.18653/v1/2020.findings-emnlp.71. URL: https://aclanthology.org/2020.findings-emnlp.71.

[14] Alec Radford y Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. En: 2018.

[15] Stephen Robertson y Hugo Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond”. En: *Found. Trends Inf. Retr.* 3.4 (abr. de 2009), págs. 333-389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: https://doi.org/10.1561/1500000019.

[16] Svitlana Vakulenko et al. “Message Passing for Complex Question Answering over Knowledge Graphs”. En: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM ’19. Beijing, China: Association for Computing Machinery, 2019, págs. 1431-1440. ISBN: 9781450369763. DOI: 10.1145/3357384.3358026. URL: https://doi.org/10.1145/3357384.3358026.

[17] World Wide Web Consortium (W3C). *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation. 2012. URL: https://www.w3.org/TR/owl2-overview/.

[18] World Wide Web Consortium (W3C). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[19] World Wide Web Consortium (W3C). *RDF Schema 1.2*. W3C First Public Working Draft. 2023. URL: https://www.w3.org/TR/rdf12-schema/.

[20] World Wide Web Consortium (W3C). *SPARQL 1.1 Query Language*. W3C Recommendation. 2013. URL: https://www.w3.org/TR/sparql11-query/.

[21] Ikuya Yamada et al. *Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia*. 2020. arXiv: 1812.06280 [cs.CL].

[22] Chengxiang Zhai y John Lafferty. “A study of smoothing methods for language models applied to information retrieval”. English (US). En: *ACM Transactions on Information Systems* 22.2 (abr. de 2004), págs. 179-214. ISSN: 1046-8188. DOI: 10.1145/984321.984322.

[23] Nikita Zhiltsov, Alexander Kotov y Fedor Nikolaev. “Fielded Sequential Dependence Model for Ad-Hoc Entity Retrieval in the Web of Data”. En: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’15. Santiago, Chile: Association for Computing Machinery, 2015, págs. 253-262. ISBN: 9781450336215. DOI: 10.1145/2766462.2767756. URL: https://doi.org/10.1145/2766462.2767756.

A. Configuraciones y resultados de DBpedia-Entity

A continuación se muestra la configuración del índice usada por los sistemas de *DBpedia-Entity*, en la Tabla 4, así como sus resultados para el conjunto de pruebas en la Tabla 5, que incluye los obtenidos por los *LLM* actuales.

El primero de los sistemas, denominado A, se ha implementado en el sistema de recuperación de información *Galago*[30], el cual ha ofrecido los mejores resultados, mientras que el segundo, B, se ha implementado en *ElasticSearch*.

Campo	Descripción	Predicados
Names	Nombres de la entidad	<foaf:name>, <dbp:name>, <foaf:givenName>, <foaf:surname>, <dbp:officialName>, <dbp:fullname>, <dbp:nativeName>, <dbp:birthName>, <dbo:birthName>, <dbp:nickname>, <dbp:showName>, <dbp:shipName>, <dbp:clubname>, <dbp:unitName>, <dbp:otherName>, <dbo:formerName>, <dbp:birthname>, <rdfs:label>, <dbp:alternativeNames>, <dbp:otherNames>, <dbp:names>
Categories	Tipos asignados a la entidad	<dcterms:subject>
Similar entity names	Nombres alternativos asignados a la entidad	!<dbo:wikiPageRedirects>, !<dbo:wikiPageDisambiguates>, <dbo:wikiPageWikiLinkText> ! indica el sentido inverso (tripletas $< o, p, s >$, donde s aparece como objeto)
Attributes	Atributos literales de la entidad	Todos los $p \in < s, p, o >$ donde o es un valor literal y p no se encuentra en ninguno de los otros campos, o en una lista de predicados excluidos. Si p tiene como prefijo a <dbp:.*>, se añade la representación textual de p también.
Related entity names	Relaciones de la entidad	Equivalente al campo anterior, pero para casos donde o sea una URI
CatchAll	Campo especial general	Todos los contenidos de los campos anteriores, sin eliminar valores duplicados.

Tabla 4: Esquema del índice utilizado por los dos sistemas usados para evaluar *DBpedia-Entity*, donde A no utiliza el campo *catchAll* (las consultas de modelos como *BM25* en *Galago* se realizan implícitamente sobre todo el documento, sin necesitar establecer un campo para ello). Se puede observar cómo únicamente se han separado 25 predicados, de los 61.828 que posee el grafo de la *DBpedia*, en campos separados, mientras que el resto se encuentran agrupados o bien en el campo de relaciones o en el de atributos.

Adaptada de: https://iai-group.github.io/DBpedia-Entity/index_details.html

[30] https://www.lemurproject.org/galago.php

Modelo	Tipo	Sistema	SemSearch ES		INEX-LD		ListSearch		QALD-2		Total	
			@10	**@100**	**@10**	**@100**	**@10**	**@100**	**@10**	**@100**	**@10**	**@100**
BM25	Clásico	A	0.2497	0.4110	0.0277	0.3612	0.2199	0.3302	0.2751	0.3366	0.2558	0.3582
PRMS	Clásico	B	0.5340	0.6108	0.3590	0.4295	0.3684	0.4436	0.3151	0.4026	0.3905	0.4688
MLM-all	Clásico	B	0.5528	0.6247	0.3752	0.4493	0.3712	0.4577	0.3249	0.4208	0.4021	0.4852
LM	Clásico	B	0.5555	0.6475	0.3999	0.4745	0.3925	0.4723	0.3412	0.4338	0.4182	0.5036
SDM	Clásico	A	0.5535	0.6672	0.4030	0.4911	0.3961	0.4900	0.3390	0.4274	0.4185	0.5143
LM+ELR	Comb.	A	0.5554	0.6469	0.4040	0.4816	0.3992	0.4845	0.3491	0.4383	0.4230	0.5093
SDM+ELR	Comb.	B	0.5548	0.6680	0.4104	0.4988	0.4123	0.4992	0.3446	0.4363	0.4261	0.5211
FSDM	Comb.	A	0.6521	0.7220	0.4214	0.5043	0.4196	0.4952	0.3401	0.4358	0.4524	0.5342
FSDM+ELR	Comb.	A	**0.6563**	**0.7257**	0.4354	0.5134	0.4220	0.4985	0.3468	0.4456	0.4590	0.5408
MLM-CA	*LTR*	A	0.6247	0.6854	0.4029	0.4796	0.4021	0.4786	0.3365	0.4301	0.4365	0.5143
BM25-CA	*LTR*	A	0.5858	0.6883	0.4120	0.5050	0.4220	**0.5142**	0.3566	0.4426	0.4399	0.5329
BM25F-CA	*LTR*	A	0.6281	0.7200	**0.4394**	**0.5296**	**0.4252**	0.5106	**0.3689**	**0.4614**	**0.4605**	**0.5505**
monoBERT	*LLM*	-	0.6430	0.7360	**0.4860**	**0.5640**	0.4930	0.5540	0.4490	0.5250	0.5150	0.5910
EM-BERT	*LLM*	-	**0.6640**	**0.7440**	0.4790	0.5610	**0.5440**	**0.5790**	**0.4830**	**0.5430**	**0.5410**	**0.6040**

Tabla 5: Resultados obtenidos con los dos sistemas utilizados para evaluar *DBpedia-Entity*. Los mejores resultados obtenidos, exceptuando los modelos de lenguaje, han sido los del modelo de *BM25F-CA*, en el que se aplica aprendizaje supervisado para optimizar sus parámetros internos y los pesos a asignar a cada campo. Los modelos de tipo *Comb.* (combinación) son mezclas de modelos clásicos o ayudados por criterios adicionales, como el *Entity Linking* [6], y los de tipo *LLM* (*Large Language Models*) son los modelos de lenguaje modernos.

Adaptada de: https://iai-group.github.io/DBpedia-Entity y los mejores resultados de los modelos de lenguaje *monoBERT* y *EM-BERT* [3].

A.1. Evaluación de configuraciones del sistema

A continuación se detallan las configuraciones analizadas durante el desarrollo del sistema.

A.1.1. Estructura del índice de ElasticSearch

Independientemente del esquema de índice generado por *Knowgly*, es posible definir subcampos a cada uno de los campos definidos por dicho esquema. Esta configuración adicional, que depende completamente del sistema de recuperación de información final a utilizar, permite almacenar y realizar búsquedas sobre los documentos en el índice bajo diferentes representaciones textuales. En el índice final de *ElasticSearch* utilizado, hemos generado y evaluado las siguientes representaciones:

- ***Text***: Este es el formato por defecto, que consiste en cadenas de texto sin ninguna representación especial. Todos los campos principales dictados por *Knowgly* siguen este formato.
- ***N-gramas* a nivel de palabra**: Los *n-gramas* a nivel de palabra consisten en generar un índice invertido basado en secuencias de palabras de longitud n. De esta forma, un índice de bigramas contendrá los documentos asociados a cada par de términos que hayan aparecido consecutivamente en la colección. Esto permite realizar consultas de frase, de tal forma que la consulta "futbolistas Italianos famosos" se procesará como una consulta de "futbolistas Italianos" e "Italianos famosos", capturando así mejor las dependencias posicionales de los términos, frente a la realización de una consulta de cada término por separado.

  En el caso de *Knowgly*, la búsqueda sobre índices de este estilo, en combinación con otros subcampos, permite potenciar resultados que mencionen el nombre de una entidad en una consulta, como "*Michael Schumacher*", de manera perfecta. Sin embargo, al utilizar el conjunto de evaluación, *DBpedia-Entity*, hemos observado que los *n-gramas* no han mejorado los resultados de las consultas, sino que los han empeorado ligeramente, por lo que hemos optado por no utilizarlos de momento.
- ***Keywords***: Los campos de palabras clave o *keyword* establecen un método estricto de emparejamiento de términos de la consulta con los del documento, de tal forma que un término de dicha consulta debe coincidir exactamente con uno del documento para generar un *match*.

  Este tipo de campo es útil a la hora de contrarrestar las desventajas causadas por el uso de técnicas de procesamiento del texto de documentos, como el *stemming*, que pueden provocar falsos positivos al eliminar sufijos de los términos.

Durante la evaluación automática hemos utilizado únicamente la representación textual básica, para así establecer una base común para todos los modelos de recuperación y de consulta. Una vez obtenidos los mejores resultados, hemos enriquecido el índice y las consultas con la representación de *keywords*, que ha mejorado ligeramente los resultados.

A.1.2. Técnicas de tratamiento de la entrada

Este aspecto, que define las técnicas a utilizar para transformar el texto de los documentos y consultas, es la configuración externa a *Knowgly* que más impacto tiene sobre los resultados finales, al poder mejorar considerablemente la precisión y la exhaustividad.

ElasticSearch permite configurar estos aspectos mediante el uso de analizadores, que son una abstracción sobre una serie de filtros a aplicar sobre el texto secuencialmente. Este aspecto se ha configurado mediante la definición de analizadores personalizados, como parte de la definición del índice general de *ElasticSearch* que *Knowgly* utiliza durante su creación. Los filtros utilizados son los siguientes:

- **Filtro de minúsculas**: El filtro de minúsculas genera una normalización completa del texto de documentos y consultas, de tal forma que consultas como "Formula 1" y "formula 1" generan los mismos resultados. No aplicar este filtro puede provocar que una de esas dos consultas no genere buenos resultados, al tratar "Formula" y "formula" como términos independientes, pudiendo aparecer uno de ellos con menor frecuencia o en contextos completamente diferentes.

- **Filtro de *stopwords***: La eliminación de *stopwords* o palabras vacías, como son los artículos, pronombres o preposiciones, elimina una gran cantidad de ruido en las consultas. Esto se debe a que dichas palabras aparecen con mucha frecuencia en el texto, y generar consultas para ellas siempre va a pesar documentos no relevantes.

- ***Stemming***: El *stemming* consiste en eliminar sufijos de palabras, de tal forma que se mantienen únicamente sus raíces. De esta forma, verbos como "walking" y "walked" pasan a ser en ambos casos "walk". Este filtro, aunque a simple vista puede parecer aumentar el número de posibles falsos positivos, genera un aumento considerable del *recall*. Esto es debido a que permite converger palabras con un mismo significado pero diferentes sufijos, como singulares y plurales o diferentes tiempos verbales, en una sola raíz.

  El uso de este filtro junto a la representación basada en *keywords*, que contrarresta sus inconvenientes, ha mejorado considerablemente los resultados finales.

Durante la evaluación automática hemos aplicado todos estos filtros a la representación básica de *text*, así como un filtro de paso a minúsculas al utilizar posteriormente la representación de *keyword*, para así contrasrrestar con este campo falsos positivos derivados de *stemming* y el filtrado de palabras vacías. Esto ha conseguido mejorar considerablemente los resultados.

A.1.3. Modelos de recuperación utilizados

Knowgly hace uso de los modelos de *BM25*, *BM25F* [15] y *LM* [22] a la hora de realizar consultas sobre *ElasticSearch*.

Durante las pruebas automáticas, hemos utilizado *BM25* y *BM25F* como modelos de recuperación en las consultas y, una vez obtenidas las mejores configuraciones de índice, hemos probado a repetirlas cambiando únicamente el modelo de recuperación subyacente a *LM*. Esto lo hemos realizado así al haber comprobado

empíricamente que *LM* mejora proporcionalmente los resultados, sin haber encontrado ningún empeoramiento. Con esto, evitamos duplicar el conjunto de pruebas a ejecutar.

En ambos casos, el ajuste de sus parámetros es esencial a la hora de configurar sistemas de recuperación de información, debido a las grandes mejores en precisión y exhaustividad que ofrecen una vez adaptados a la colección.

Ajuste de *BM25* El modelo de *BM25* define dos parámetros libres. El primero de ellos, $k1$, define la saturación de la frecuencia de un término (*term frequency*)[31] en el documento. Esta saturación se refiere al efecto que *BM25* produce en el aumento en la puntuación final en función de la frecuencia del término, que pasa de ser lineal a ser asintótico, tal que un término que aparece 20 veces en un documento tiene casi el mismo impacto que si apareciera 1000 veces. Un valor bajo de $k1$ provoca que múltiples apariciones de un término saturen la curva mucho más rápido, mientras que uno alto favorece más las múltiples apariciones. En el caso de *Knowgly*, el valor de $k1$ se ha reducido de su valor por defecto de 1.2 a 0.8, bajo la intuición de que no queremos favorecer excesivamente a entidades que cuenten con muchas relaciones con otra, y por tanto cuenten con un gran número de menciones a ella.

Por otro lado, el parámetro b controla el efecto de la normalización de las puntuaciones por la longitud relativa del documento frente al resto. De esta forma, se consigue que un documento largo sea penalizado frente a otro de menor longitud, bajo la suposición de que un documento más largo tiene mayor probabilidad de contener los términos de una consulta, sin tener por qué ser relevante. En nuestro caso, hemos reducido su valor por defecto de 0.75 a 0.5, ya que no queremos penalizar excesivamente entidades mucho mejor descritas, en cuyo caso es más probable que sean relevantes.

Ajuste de *LM* El modelo de *LM*, utilizando la variante de *Dirichlet Smoothing*, define un parámetro libre μ, que sirve como un factor que aumenta proporcionalmente el número de ocurrencias de un término del documento, aumentando también su longitud en el proceso. Este parámetro se puede tomar como el punto de partida para la media de apariciones de cada término en un documento.

En nuestro caso, hemos disminuido considerablemente su valor por defecto de 2000 a 150, bajo la suposición de que la longitud de nuestros documentos no va a ser tan elevada, y por tanto las frecuencias de los términos en ellos no necesitan ser tan potenciadas, ya que una única mención de un término en una entidad ya puede ser relevante.

A.1.4. Modelos de consulta de ElasticSearch

ElasticSearch ofrece abstracciones sobre la mayoría de los modelos de recuperación implementados por *Lucene*[32], como *BM25* o *LM*, así como varios modelos propios[33]. En función de estos modelos, es posible describir consultas con diferentes comportamientos, como consultas *booleanas*, consultas de *BM25F* o mezclas de

[31] https://en.wikipedia.org/wiki/Tf-idf

[32] https://lucene.apache.org/core/7_0_1/core/org/apache/lucene/search/similarities/Similarity.html

[33] https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html

consultas básicas que utilicen cualquier modelo, con un gran número de condiciones configurables.

Modelos de consulta analizados En el contexto de *Knowgly*, necesitamos realizar consultas sobre los múltiples campos del índice generado, así como consultas básicas sobre el campo *catchAll* para comparar sus comportamientos. Debido a esto, hemos evaluado el uso de las siguientes consultas:

- ***MatchQuery***: Las *MatchQuery* son consultas básicas sobre un único campo y analizador, que utilizan un modelo como *BM25* o *LM* de forma subyacente. Este tipo de consulta se ha utilizado a la hora de ejecutar una consulta básica sobre el campo *catchAll*, sin utilizar representaciones textuales adicionales, o en consultas compuestas.
- ***MultiMatchQuery***: Las *MultiMatchQuery* son el equivalente a *MatchQuery* para varios campos, pudiendo aplicar una consulta a un número indeterminado de campos con cualquier representación, así como asignarles pesos. Estas a su vez se dividen en tres tipos, con comportamientos muy diferentes:
  - ***most_fields***: Ejecuta la consulta en cada campo de forma separada, de tal forma que cada campo tiene unos *Term Frequency* (*TF*) e *Inverse Document Frequency* (*IDF*)[34] propios.

    Este tipo de consulta no es muy deseable debido a la alta probabilidad de generar falsos positivos. Esto se debe a la posibilidad de que se pese demasiado un término de la consulta en un campo espurio en el que aparezca con una frecuencia muy baja, y por tanto se prioritice.

    No obstante, hemos decidido optar una variante a esta consulta para la experimentación mucho más controlable, de tal forma que somos capaces de controlar y verificar el efecto de los pesos sobre cada campo.
  - ***best_fields***: Equivalente a *most_fields*, con la diferencia de que devuelve la puntuación del mejor campo, desechando las del resto.

    Este tipo de consulta rompe con nuestro esquema de pesos al ignorar las contribuciones del resto de campos, así que no se ha utilizado.
  - ***cross_fields***: Combina todos los campos en uno solo y ejecuta la consulta sobre él, término a término. Sigue una estrategia similar a la de *BM25F*, con un *IDF* homogéneo para todos los campos, pero utiliza un *TF* global también, bajo la suposición de que se quieren evitar los problemas derivados de *most_fields*. Esto hace que se asemeje a una consulta *MatchQuery*, tras fusionar de forma lógica todos los campos.

    Este modelo de consulta fue utilizado inicialmente para comparar el comportamiento de una consulta sobre *CatchAll* frente a una sobre los campos fusionados. Sin embargo, su modelo de puntuación no sigue una base teórica analizable y es opaco a la hora de analizar los resultados de las consultas, ofreciendo resultados ligeramente mejores a *catchAll* en algunos casos y mucho peores en otros. Esto se debe a que tiene comportamientos adicionales como elecciones internas de mejores puntuaciones, en vez de sumas. Adicionalmente, los pesos a los campos, aunque configurables, no afectan al resultado, por lo que también choca con nuestro

[34] https://en.wikipedia.org/wiki/Tf-idf

modelo.

- ***CombinedFieldsQuery***: Este modelo de consulta implementa el modelo *BM25F* con diferentes pesos sobre un número determinado de campos, con ciertas restricciones como la imposibilidad de mezclar campos de diferentes representaciones, como *text* y *ngramas*, y de asignar pesos con valores menores a 1.0. A diferencia de una *MultiMatchQuery*, el valor de *IDF* es global para todos los campos, mientras que *TF* es local a cada uno, de tal forma que el peso multiplica la *term frequency* y aumenta la longitud de dicho campo.

  Como gran desventaja, el hecho de que *ElasticSearch* imponga que los pesos de los campos deben ser mayores a 1.0 choca con nuestra estrategia de pesos óptima, basada en penalizaciones. Aunque es posible escalar los pesos, de tal forma que un esquema de pesos [1.0, 0.1, 0.05] pasa a ser [20.0, 2.0, 1.0], el efecto sobre las puntuaciones finales no es proporcional. Por otro lado, la optimización del modelo es compleja, al no tener la posibilidad de mejorar sus resultados con diferentes representaciones textuales.

- ***BooleanQuery***: El modelo de *BooleanQuery* o *BoolQuery* permite construir una consulta booleana, que consiste en la combinación de varias consultas mediante operadores booleanos, como *and*, *and not* u *or*, o mediante filtros.

  El resultado de este tipo de consultas es la suma de las puntuaciones devueltas por cada una de las consultas internas, que pueden ser de cualquier tipo, con los pesos multiplicando dichos resultados.

  Estas consultas sirven como el bloque básico de construcción de cualquier tipo de consulta propio, y en nuestro caso las hemos usado para construir una consulta alternativa a *CombinedFieldsQuery*.

Modelos de consulta utilizados Una vez excluidos los modelos con comportamientos no deseados, hemos decidido utilizar los siguientes modelos:

- ***MatchQuery***: Este modelo lo hemos utilizado para realizar consultas básicas sobre *catchAll*.

- ***CombinedFieldsQuery***: Esta consulta se ha tomado como la implementación base de *BM25F*, de tal forma que la aplicamos sobre los campos generados por *Knowgly*, pesados por su importancia, junto al *catchAll*, que produce un efecto normalizador.

- ***BooleanQuery***: Hemos construido una consulta propia mediante *BooleanQuery* equivalente a *most_fields*, con el objetivo de evitar las desventajas de *CombinedFieldsQuery*.

  Esta consulta está compuesta de una *BooleanQuery* por cada campo del índice, que engloba y suma los resultados de consultas *MatchQuery* sobre dicho campo y todos sus subcampos, que contendrán representaciones textuales o analizadores diferentes. Cada una de estas consultas booleanas será pesada por el peso asociado a su campo, de tal forma que sus puntuaciones se multiplican directamente por dicho valor. Estas consultas booleanas se agrupan a su vez en una *BooleanQuery* global, que únicamente suma los resultados de todas las *BooleanQuery* asociadas a cada campo.

Del mismo modo que en *CombinedFieldsQuery*, también hemos añadido una subconsulta sobre el *catchAll*, únicamente en la *BooleanQuery* global, que produce también un efecto normalizador.

Aunque inicialmente parecía contraintuitivo, debido a que los *IDF* y *TF* son locales a cada campo y pueden provocar los mismos problemas que las consultas *most_fields*, el hecho de haberlas pesado con penalizaciones ha mitigado dichos comportamientos, resultando ser la consulta con los mejores resultados.

Esta consulta también la hemos utilizado sobre únicamente el campo de *catchAll*, con el fin de poder establecer una comparativa justa frente al uso de nuestros campos. De esta forma, permitimos evaluar las consultas sobre *catchAll* usando las mismas representaciones textuales y analizadores adicionales de otros subcampos.

A.1.5. Conversión de entidades a contenido textual

El indexador de entidades, que se encarga de generar una representación textual de una entidad extraída para generar un documento virtual, permite configurar los siguientes aspectos relativos a la generación de dicho contenido:

- **Conversión de *URIs* a texto**: Un aspecto crucial a decidir es el método de conversión de *URIs* a texto. Esto se puede realizar sustituyendo caracteres por espacios a partir la propia *URI*, tras eliminar el dominio. De esta forma, una entidad como http://dbpedia.org/resource/Michael_Schumacher se transformaría directamente a *Michael Schumacher*. Otra opción más exacta es consultar los predicados *rdfs:label* que puedan tener asociadas las entidades y predicados, que indica la etiqueta de la *URI* en lenguaje natural, desambiguada en múltiples lenguajes.

  Ambas opciones se han utilizado en los dos sistemas originales de *DBpedia-Entity*, y en el caso de *Knowgly* hemos optado por utilizar una estrategia híbrida. De esta forma, las *URIs* se intentan resolver primero mediante su predicado *rdfs:label*, y si no es posible se aplica la primera aproximación mediante la sustitución de caracteres.

- **Contenido textual de predicados**: Otra de las configuraciones más relevantes es la forma en la que se introduce el contenido de relaciones en un campo. Una primera opción es, dado un predicado p con un conjunto de relaciones $\{< p, o_1 >, < p, o_2 >, \ \dots \ , < p, o_n >\}$, añadir el contenido de sus objetos secuencialmente. De esta forma, el campo consistiría en el contenido textual de $\{o_1, o_2, \ \dots \ , o_n\}$ separados por espacios, junto al contenido de otras relaciones.

  Otra opción es añadir también el contenido textual de los predicados junto a esa secuencia. Esto se debe realizar cuidadosamente, ya que incluir predicados con contenido textual relevante, como los que indican profesiones, pueden causar confusiones. Por ello, al utilizar esta opción, se añade el contenido de dichos predicados una única vez antes de la secuencia de objetos, tal que el contenido en el campo sería la representación textual de $\{p, o_1, o_2, \ \dots \ , o_n\}$.

  Esto es idéntico a la estrategia utilizada por los sistemas de *DBpedia-Entity*, que se aplicaba en función de los campos, no incluyéndolos así en campos que

se sabe que contienen únicamente predicados de nombre. Como en nuestro caso no podemos suponer qué predicados van a ser asociados a un campo, esta estrategia ha sido aplicada en todos ellos.

El uso de cualquiera de estas dos aproximaciones no ha producido cambios significativos en los resultados, aunque hemos optado añadir el texto de los predicados en la configuración por defecto. Esto espera poder ayudar en consultas muy específicas, en las cuales algunos de sus términos coinciden con el texto de predicados relevantes a una entidad. Por ejemplo, si se aplica esta técnica, la consulta "*Formula 1 record drivers*" puede asignar más peso a entidades que posean el predicado <*dbp:recordDriver*>, cuya representación textual es directamente *Record driver*.

- **Exclusión de contenido textual**: El contenido textual de ciertos predicados, como los de *rdfs*, cuyas etiquetas serían, por ejemplo, *class* o *domain*, no es relevante a la hora de describir una entidad. Esto se ha solucionado permitiendo configurar prefijos, de tal forma que solo se añade el texto de los predicados cuya *URI* está contenida en alguno de los prefijos.

  Esto permite añadir únicamente predicados del ámbito del grafo de conocimiento específico, como http://dbpedia.org/property/∗ en el caso de la *DBpedia*. Esta estrategia ha sido también utilizada en los sistemas de *DBpedia-Entity*, y se ha replicado en *Knowgly*.

- **Exclusión de entidades a indexar**: Por último, también existe la opción de establecer un conjunto de predicados que una entidad debe tener para ser indexada, permitiendo así establecer un filtro de entidades espurias, como sujetos sin relaciones o atributos.

  Esto ha sido utilizado también en los sistemas de *DBpedia-Entity*, de tal forma que solamente se indexan entidades que tengan los predicados <*rdfs:label*> y <*rdfs:comment*>. *Knowgly* también sigue esta estrategia, para poder así partir de un mismo conjunto de documentos para la evaluación.

A.1.6. Agregadores de métricas

Los agregadores de métricas poseen una configuración extensiva, relativa a la creación de los esquemas de documentos virtuales y su estructuración en campos.

- **Pesos a asignar a cada campo**: Los pesos asignados a cada campo corresponden directamente a los *boosts* a realizar en tiempo de ejecución de consultas, de tal forma que los campos con los predicados más importantes tienen un peso mayor y, por tanto, un *boost* mayor.

  Todos los pesos son crecientes de menor a mayor importancia, debiendo evitar asignar pesos extremadamente altos o no lineales, y permiten penalizar campos con valores en el rango [0.0, 1.0]. En la Sección A.1.7 de los Anexos se pueden encontrar las estrategias de asignación de pesos utilizadas durante la evaluación.

- **Número de campos a utilizar**: El número de campos, y por tanto el número de clústeres a utilizar en *KMeans++*, no debe ser excesivamente pequeño o grande. Un número pequeño de clústeres puede causar que predicados de

alta importancia se mezclen con predicados espurios o de baja importancia, provocando falsos positivos.

Por otro lado, con la estrategia de indexación actual, en la cual generamos un esquema de índice global, un número excesivo de clústeres puede causar que haya campos con pocos predicados muy específicos asignados a ellos, estando como consecuencia vacíos en la mayoría de entidades. Adicionalmente, la asignación de pesos es mucho más compleja de establecer con mayores números de campos, al ser una tarea manual.

Estos problemas se podrían mitigar al generar un esquema de indexación basado en tipos, con plantillas de documento virtual a nivel de cada entidad, que es una tarea pendiente de analizar, tal y como se encuentra detallado en la Sección 6.2. El uso de esta aproximación favorecerá que todos los campos contengan contenido textual, al realizarse la clusterización únicamente sobre los predicados asociados a cada entidad.

Durante la evaluación del sistema, hemos probado la división en 3 y 5 campos, así como otra en 6 campos mediante la división de tipos de predicados, descrita a continuación.

- **Separar tipos de predicados**: Las ontologías *OWL* permiten realizar una distinción lógica entre predicados, definiendo las *Datatype properties*, predicados únicamente asociados a objetos literales, y las *Object properties*, predicados únicamente asociados a *URIs*.

  Los agregadores de métricas permiten clusterizar ambos tipos de predicados de forma separada, bajo la suposición de que los rangos de valores de sus importancias pueden ser diferentes según el tipo de objetos con los que se relacionan.

  Se debe observar que, al organizarse de forma separada, se duplica el número de campos a utilizar en el esquema, de tal forma que en un índice de n campos habrá en realidad $2n$ campos, con la mitad de ellos correspondiendo a un tipo de predicado y la otra mitad al otro.

A.1.7. Asignaciones de peso a los campos

Knowgly permite asignar pesos a cada uno de los campos del índice que genera, con el objetivo de permitir realizar consultas que tengan en cuenta la importancia de cada uno. A continuación se describe cada una de las estrategias utilizadas durante la evaluación.

Asignar pesos equitativos a cada campo La asignación de pesos equitativos produce un efecto similar al obtenido por un campo *catchAll*, que engloba el contenido de todos ellos. Utilizar este esquema ignora por completo sus importancias, aunque otorga el beneficio de que, en algoritmos como *BM25F*, se tengan en cuenta las longitudes de documento a nivel de cada uno de los campos. Esto puede ser beneficioso en casos donde los campos de mayor prioridad sean menos extensos, en número de términos, que el resto, en cuyo caso sus puntuaciones tendrán mayor peso.

En una clusterización en 3 campos, esta estrategia equivaldría a establecer los pesos [1.0, 1.0, 1.0].

Asignar mayores pesos a campos de mayor importancia Esta estrategia consiste en aumentar progresivamente los pesos para los campos de mayor importancia. Esto consigue tener en cuenta la importancia de los campos, a la vez que el campo de menor importancia no es penalizado.

Aunque esta es una mejor estrategia que la anterior, el modelo de *BM25F* [15] sigue siendo problemático, ya que el peso es únicamente un factor multiplicativo sobre la frecuencia de los términos en dicho campo, que a su vez aumenta la longitud de los documentos. Esto puede causar problemas en situaciones en las que las descripciones de la entidad se encuentren en el campo prioritario y las relaciones, que contienen menciones a otras entidades, en un campo de menor prioridad. Al consultar términos relativos a una entidad s, una entidad r con muchas relaciones con s, y por tanto menciones textuales de ella, podría tener la misma o mayor puntuación que la propia entidad s, que tendría apariciones de términos en su campo prioritario, pero con menor frecuencia, aun tras aplicar *boosts*.

En una clusterización en 3 campos, esta estrategia equivaldría a establecer, por ejemplo, los pesos [2.0, 1.5, 1.0].

Penalizar campos de menor importancia Esta estrategia realiza lo contrario a la anterior, de tal forma que el peso del mayor campo se mantiene en su valor original, mientras que los de los campos de menor prioridad son mucho menores o simplemente son nulos.

Esto busca evitar el problema derivado de la anterior estrategia, ya que hemos podido observar que, en el grafo de la *DBpedia* utilizado, dichas situaciones aparecen con gran frecuencia, al acumularse la gran mayoría de relaciones poco relevantes en el clúster de menor importancia. De esta forma, se puede tratar la clusterización como un filtrado de datos y relaciones espurias, que únicamente causan falsos positivos en las consultas, pesando así mucho menos, o incluso descartando, los campos de menor importancia. Este esquema es el que ha proporcionado los mejores resultados, que se analizan en profundidad en el Anexo C.3.

En una clusterización en 3 campos, esta estrategia equivaldría a establecer, por ejemplo, los pesos [1.0, 0.15, 0.10], o a anular el último campo en las búsquedas con los pesos [1.0, 0.15, 0.0].

B. Resultados de la evaluación automática

A continuación se muestran los resultados de la evaluación automática detallada en la Sección 5.2.1. Todos los resultados se han obtenido con los mismos esquemas de pesos (elegidos manualmente tras unos pocos experimentos), una representación textual básica y el modelo de recuperación de *BM25*, sin realizar ningún ajuste de sus parámetros. De esta forma, hemos conseguido identificar las diferencias entre los esquemas de índice generados por cada agregador de métricas.

El criterio de selección de la mejor clusterización ha consistido en buscar el mejor resultado de $NDCG@10$ para *BoolQuery*, que es nuestra consulta booleana propia, o *BM25F*, teniendo siempre en cuenta que debe superar en puntuación a la obtenida por *CatchAll*. De esta forma, obtenemos los casos en los que la división en campos es más efectiva que una búsqueda tradicional.

En cada una de las tablas se han resaltado los mejores resultados para nuestras consultas y *catchAll*, subrayando aquellos que se encuentren muy próximos.

B.1. Resultados para 3 clústeres

En la clusterización en 3 campos hemos podido observar que el agregador de *InfoRank*, con una puntuación de 0.4111, ha sido el que mejor comportamiento ha presentado, superando ligeramente a las devueltas por la mejor construcción del campo *CatchAll*, con una puntuación de 0.4077. Su combinación equitativa con la variante de media geométrica de las métricas de importancia estructural ha ofrecido resultados casi idénticos. Los resultados se encuentran en la Tabla 6.

Ambas puntuaciones fueron modificadas, tras añadir representaciones textuales y ajustar los modelos y el esquema de pesos, a 0.4290 y 0.4032 respectivamente, donde *CatchAll* sufrió una ligera penalización.

B.2. Resultados para 5 clústeres

En el caso de 5 campos, los mejores resultados fueron devueltos por las consultas de *BM25F* que, sin embargo, no llegaron a superar a la mejor del campo *CatchAll*, 0.4077. En este caso, optamos por elegir el mejor resultado para *BoolQuery*, para *InfoRank* de nuevo, debido a que las limitaciones de *ElasticSearch* no nos permiten añadir a representaciones textuales adicionales a *BM25F* ni realizar penalizaciones en los pesos, y por tanto no es posible mejorar tanto sus resultados. Los resultados se encuentran en la Tabla 7.

Ambas puntuaciones fueron modificadas, tras añadir representaciones textuales y ajustar los modelos y el esquema de pesos, a 0.4394 y 0.4032 respectivamente, donde *CatchAll* sufrió una ligera penalización de nuevo, y el nuestro resultó ser el mejor de todos los obtenidos.

B.3. Resultados para la separación de tipos de predicados

En el caso de la separación por tipos de predicados, el comportamiento ha sido el mismo que el del caso anterior, con ninguna de nuestras puntuaciones consiguiendo superar a las de *CatchAll*. En este caso, también optamos por elegir el resultado de *InfoRank*. Los resultados se encuentran en la Tabla 8.

Ambas puntuaciones fueron mejoradas, tras añadir representaciones textuales y ajustar los modelos y el esquema de pesos, a 0.4234 para nuestra consulta y 0.4036 para *CatchAll*.

Agregadores de métricas			**Consulta**	**Puntuación**	
Agregador de métricas 1	**Agregador de métricas 2**	**Ponderación (sobre 1)**		**@10**	**@100**
Predicate-Based Structural Importance (Geometric Mean)	-	-	BoolQuery	0.1710	0.2321
			BM25F	0.3716	0.4376
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.4079	0.4698
			BM25F	0.3599	0.4189
			CatchAll	0.3899	0.4512
		0.5	BoolQuery	0.4108	0.4735
			BM25F	0.3628	0.4221
			CatchAll	0.3905	0.4521
		0.75	BoolQuery	0.3717	0.4162
			BM25F	0.3733	0.4358
			CatchAll	0.3970	0.4591
Predicate-Based Structural Importance (Multiplication)	-	-	BoolQuery	0.3333	0.4076
			BM25F	0.3917	0.4544
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.3940	0.4548
			BM25F	0.3849	0.4445
			CatchAll	0.3978	0.4588
		0.5	BoolQuery	0.3506	0.4225
			BM25F	0.3930	0.4543
			CatchAll	0.4031	0.4628
		0.75	BoolQuery	0.3225	0.3964
			BM25F	0.3903	0.4522
			CatchAll	0.4071	0.4672
Predicate-based InfoRank	**-**	-	BoolQuery	**0.4111**	**0.4735**
			BM25F	0.3590	0.4199
			CatchAll	0.3887	0.4507

Tabla 6: Resultados de la evaluación automática para 3 clústeres, con pesos [1.0, 0.1, 0.05] (escalados a [20.0, 2.0, 1.0] en el caso de *BM25F*).

Agregadores de métricas			Consulta	Puntuación	
Agregador de métricas 1	**Agregador de métricas 2**	**Ponderación (sobre 1)**		**@10**	**@100**
Predicate-Based Structural Importance (Geometric Mean)	-	-	BoolQuery	0.2814	0.3553
			BM25F	0.3721	0.4371
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.3731	0.4368
			BM25F	0.3601	0.4196
			CatchAll	0.3878	0.4514
		0.5	BoolQuery	0.3752	0.4443
			BM25F	0.3783	0.4383
			CatchAll	0.3885	0.4512
		0.75	BoolQuery	0.2878	0.3504
			BM25F	0.3698	0.4296
			CatchAll	0.3907	0.4515
Predicate-Based Structural Importance (Multiplication)	-	-	BoolQuery	0.3185	0.3953
			BM25F	0.3736	0.4317
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.3661	0.4103
			BM25F	0.3847	0.4447
			CatchAll	0.3956	0.4557
		0.5	BoolQuery	0.3593	0.4329
			BM25F	**0.3973**	**0.4582**
			CatchAll	0.4011	0.4616
		0.75	BoolQuery	0.2728	0.3507
			BM25F	0.3854	0.4488
			CatchAll	0.4056	0.4655
Predicate-based InfoRank	**-**	-	BoolQuery	0.3762	0.4404
			BM25F	0.3736	0.4317
			CatchAll	0.3877	0.4513

Tabla 7: Resultados de la evaluación automática para 5 clústeres, con pesos $[1.0, 0.5,\ 0.15,\ 0.10,\ 0.05]$ (escalados a $[20.0,\ 10.0,\ 3.0,\ 2.0,\ 1.0]$ en el caso de *BM25F*).

Agregadores de métricas			Consulta	Puntuación	
Agregador de métricas 1	**Agregador de métricas 2**	**Ponderación (sobre 1)**		**@10**	**@100**
Predicate-Based Structural Importance (Geometric Mean)	-	-	BoolQuery	0.1514	0.2124
			BM25F	0.3716	0.4376
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.3669	0.4371
			BM25F	0.3599	0.4189
			CatchAll	0.3874	0.4494
		0.5	BoolQuery	0.3723	0.4261
			BM25F	0.3628	0.4221
			CatchAll	0.3883	0.4516
		0.75	BoolQuery	0.2381	0.3027
			BM25F	0.3733	0.4358
			CatchAll	0.3914	0.4525
Predicate-Based Structural Importance (Multiplication)	-	-	BoolQuery	0.3284	0.4013
			BM25F	0.3917	0.4544
			CatchAll	**0.4077**	**0.4689**
	Predicate-based InfoRank	0.25	BoolQuery	0.3322	0.3722
			BM25F	0.3849	0.4445
			CatchAll	0.3957	0.4557
		0.5	BoolQuery	0.2647	0.3356
			BM25F	**0.3930**	**0.4543**
			CatchAll	0.3994	0.4595
		0.75	BoolQuery	0.1652	0.2352
			BM25F	0.3903	0.4522
			CatchAll	0.4022	0.4631
Predicate-based InfoRank	**-**	-	BoolQuery	0.3792	0.4382
			BM25F	0.3590	0.4199
			CatchAll	0.3888	0.4507

Tabla 8: Resultados de la evaluación automática para 3 clústeres con separación de predicados, asignando los pesos [1.0, 0.1, 0.05] a ambos (escalados a [20.0, 2.0, 1.0] en el caso de *BM25F*).

C. Análisis de los resultados finales

C.1. Significación estadística de los resultados

Con el objetivo de probar que los resultados finales que hemos obtenido cuentan con significancia estadística[35], es decir, que son lo suficientemente diferentes unos de otros y por tanto sus mejoras son válidas, hemos utilizado la herramienta *pytrec_eval*[36] para comparar entre pares los resultados finales. Este procedimiento, en el campo de la recuperación de la información, se suele realizar mediante una comparación de cada uno de los resultados frente a los más bajos.

Fijándonos en la medida *NDCG*@10, hemos obtenido los *p*-valores de la comparación entre nuestros mejores resultados para 3 y 5 clústeres respecto a los de 3 clústeres con división por tipos de predicado, que han proporcionado la menor puntuación. Esto se muestra en la Tabla 9. Partiendo de la hipótesis nula de que las distribuciones estadísticas formadas por los resultados son idénticas, podemos concluir que los mejores resultados, para 5 clústeres, son lo suficientemente diferentes como para rechazarla y por tanto representan una mejora real del sistema. Por otro lado, la diferencia entre los resultados para 3 clústeres no es muy clara al tener un *p*-valor elevado. Esto se puede atribuir al poco peso que tienen las relaciones a la hora de realizar búsquedas en la *DBpedia*, cuya división no produce mejoras significativas. Este comportamiento se detalla en el Anexo C.3.

Por otro lado, hemos realizado la misma comparación frente a los resultados más bajos de *DBpedia-Entity*, que corresponden al uso de *BM25* sin ajuste de parámetros. En este caso, todos nuestros resultados son significativamente diferentes al tener *p*-valores despreciables, como se puede observar en la Tabla 10.

	3 clústeres (con división de tipos de predicado)
3 clústeres	0.3089
5 clústeres	0.0032

Tabla 9: *p*-valores obtenidos al comparar nuestros resultados.

	BM25 (DBpedia-Entity)
3 clústeres	4.05e-37
5 clústeres	1.65e-37
3 clústeres (con división de tipos de predicado)	4.35e-36

Tabla 10: *p*-valores obtenidos para nuestros resultados frente a *DBpedia-Entity*.

[35] https://es.wikipedia.org/wiki/Significación_estadística
[36] https://github.com/cvangysel/pytrec_eval

C.2. Análisis de los tipos de consulta

Un aspecto crucial a considerar en la comparación frente a los otros sistemas, además de la puntuación final, es el comportamiento de *Knowgly* frente a cada uno de los tipos de consulta existentes en *DBpedia-Entity*. Esto se puede observar, para la configuración de 5 clústeres con los mejores resultados, en la Figura 13, siendo las consultas las siguientes:

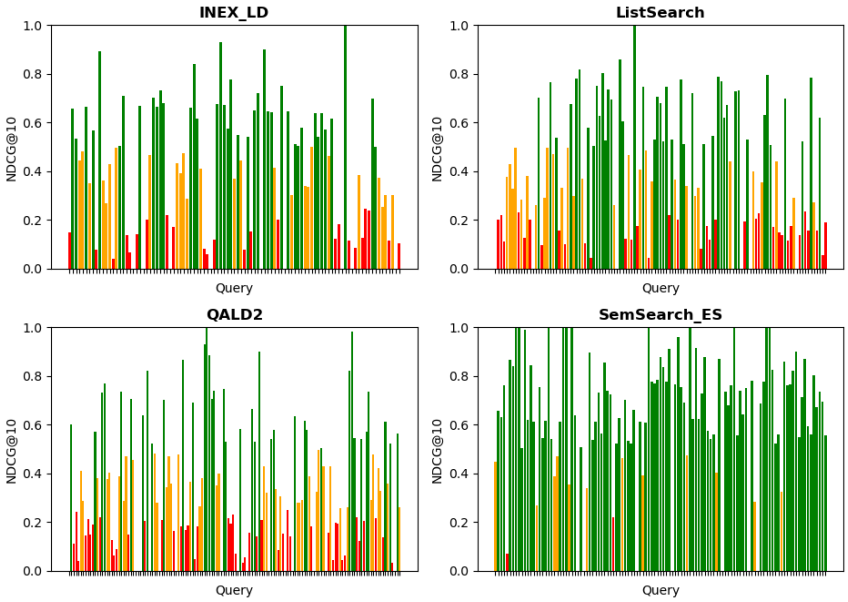

Figura 13: Resultados para la métrica $NDCG@10$ sobre cada familia de consultas de *DBpedia-Entity*, pertenecientes a la configuración de 5 clústeres con los mejores resultados.

- ***SemSearch_ES***: Estas consultas, que indican nombres de entidades concretas, son las que mejor trata nuestro sistema debido a su sencillez. Obtener un buen resultado en este conjunto es primordial, al ser un modelo de consulta muy común en los motores de búsqueda web.

  En nuestro caso, las puntuaciones obtenidas para este conjunto a lo largo de las tres configuraciones son muy dispares, aunque esto es algo que también ocurre en los sistemas originales, siendo todas ellas aun así muy buenos resultados.

  En la clusterización de 5 campos, que es el sistema con la mejor puntuación global, hemos alcanzado una puntuación de 0.6458. Esto supera a todos los métodos tradicionales y se aproxima, por muy poca diferencia, a los métodos combinados o entrenados por aprendizaje supervisado, incluyendo además a los *LLM*, cuyas mejores puntuaciones son 0.6563 y 0.6640 respectivamente.

- ***INEX-LD***: Estas consultas de palabras clave, que pueden referirse a una o varias entidades explícita o implícitamente, son también uno de los puntos fuertes de nuestro sistema. En este caso, hemos obtenido una puntuación de 0.4026 con la configuración de 5 clústeres, y de 0.4082 con la división de tipos

de predicado, frente a una puntuación de 0.4394 para el modelo de *BM25F* entrenado y un 0.4860 de los *LLM*.

Haber conseguido una puntuación menor a la de *SemSearch_ES* indica que estas consultas causan confusiones en ciertas situaciones, que es exactamente lo que ocurre. En algunos casos, los resultados son aceptables o perfectos, como en la consulta "*Bob Ricker Executive Director the latest front group for the anti-gun movement*", que es extremadamente concreta, mientras que en otros no se devuelve ningún resultado relevante, como en las consultas "*vietnam food recipes*" o "*bicycle benefits health*", que son consultas muy abiertas o incluso de poca relevancia para la búsqueda de entidades, como en el último caso.

- ***QALD2***: Este conjunto, compuesto por preguntas formuladas en lenguaje natural, es en este caso el punto más problemático de nuestro sistema. Esto se debe a que *Knowgly* no está orientado a la tarea de *Question Answering*, y por tanto va a presentar resultados no óptimos. Estos resultados, sin embargo, son muy cercanos a los obtenidos por los sistemas de *DBpedia-Entity*, debido a que ellos tampoco están preparados para esta tarea.

  Los *LLMs*, por otro lado, poseen una ventaja inherente ante este tipo de consultas, aunque la inclusión de un sistema de *Question Answering* en *Knowgly*, tal y como se plantea en la Sección 6.2, podría mejorar considerablemente nuestros resultados.

- ***ListSearch***: Este último conjunto, compuesto por consultas variadas referentes a grupos de entidades, presenta un comportamiento inverso a *INEX-LD*, con malos resultados para consultas largas como "*Operating systems to which Steve Jobs related*" y resultados perfectos para consultas más cortas como "*Paul Auster novels*".

  Nuestros resultados, en este caso, son ligeramente mejores a los de los métodos tradicionales, con un 0.3996 frente al 0.3961 de *SDM*, aunque no se alejan demasiado de los del mejor modelo, *BM25F-CA*, con 0.4252. Los *LLM* cuentan también con una ventaja inherente del mismo modo que con *QALD2*, al encontrarse formuladas muchas de ellas en lenguaje natural.

C.3. División en clústeres y asignación de pesos

Durante la experimentación final realizada en la Sección 5.2, hemos podido verificar que los predicados con mayor informatividad y relevancia para las entidades, que incluyen la mayoría de los establecidos manualmente en el índice de los sistemas de *DBpedia-Entity* (presentados anteriormente en la Tabla 4), se sitúan en los clústeres de mayor importancia. Por otro lado, los predicados espurios tienden a agruparse en el clúster de menor importancia, de tal forma que el número de predicados asignados a un clúster es inversamente proporcional a su importancia. Esto es lo mismo que se ha podido observar anteriormente en el ejemplo de clusterización de la Figura 9, cuyos campos contaban con una asignación de 347, 3982 y 57499 predicados, respectivamente.

Este comportamiento se ha tenido que tener en cuenta especialmente a la hora de asignar los pesos a cada campo: esencialmente, hemos conseguido realizar un filtro implícito de predicados espurios o poco informativos, a los que debemos asignar pesos mucho menores o incluso suprimirlos. Un ejemplo de este comportamiento se

encuentra en la Figura 14, en la cual las puntuaciones obtenidas son inversamente proporcionales a los pesos asignados a los campos de menor prioridad.

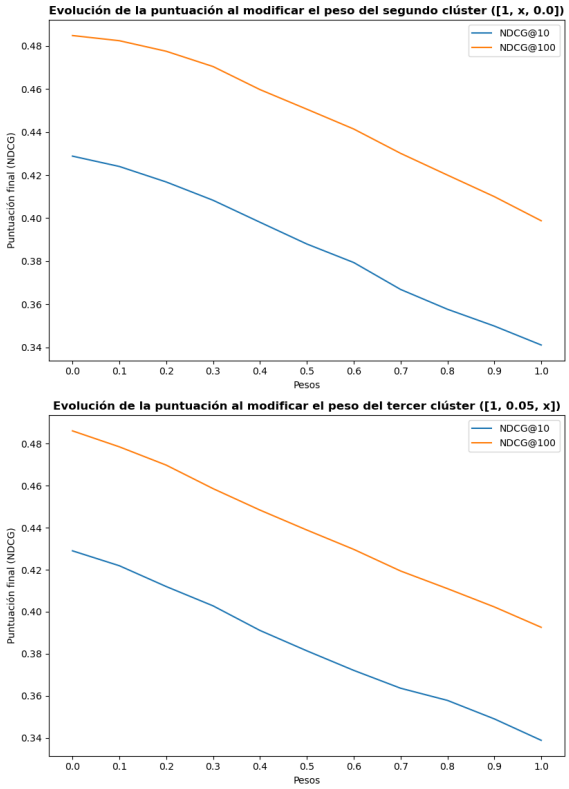

Figura 14: Empeoramiento de las puntuaciones de $NDCG@10$ y $NDCG@100$ al aumentar el valor del peso en los clústeres de menor prioridad, de forma individual. Se ha realizado sobre la clusterización en 3 campos sin separación de tipos de predicados, utilizando la *BoolQuery* con *BM25*.

C.4. Diferencias entre los generadores de métricas

Durante la evaluación automatizada realizada en la Sección 5.2, hemos podido observar que los mejores resultados se han obtenido con el agregador de métricas de *Predicate-based InfoRank*. No obstante, muchas combinaciones de agregadores de métricas han ofrecido puntuaciones muy cercanas a ella, como se puede observar en el Anexo B.

Este comportamiento se puede atribuir al papel que tienen las *Datatype Properties* generales como <*rdfs:label*> o <*rdfs:comment*>, que contienen los nombres y descripciones íntegras de las entidades, en *DBpedia-Entity*. Estos predicados se encuentran asociados a la gran mayoría de las entidades, y por tanto las métricas de *InfoRank* las pesan mucho mejor.

Por otro lado, las métricas de importancia tienden a valorar mucho más un conjunto reducido de predicados más específicos para ciertos tipos, como <*dbo:birthDate*>

o <*foaf:surname*>. Situar estos predicados por encima de otros que contienen descripciones completas de las entidades, aunque contengan también un gran valor informativo, incurre en resultados ligeramente peores en las búsquedas. Este inconveniente se ha suavizado a la hora de ponderar ambos agregadores de métricas a la vez, pero no han conseguido superar a *InfoRank*.

Esta situación, sin embargo, puede no aparecer en otros grafos en los que las *Object Properties* sean mucho más informativas de cara a la búsqueda de entidades, en cuyo caso las métricas de importancia las ordenarían mejor al pesarlas en función de la jerarquía de clases. De cara a futuro, tal y como se detalla en la Sección 6.2, vamos a utilizar un sistema de generación de esquemas de indexación basado en los tipos a nivel de cada entidad, algo que solo es posible con las métricas de importancia estructural y que previsiblemente mejorará los resultados. Por otro lado, también vamos a comprobar esta hipótesis utilizando el sistema con distintos grafos de conocimiento.

C.5. Comportamiento de los modelos de recuperación

Otro aspecto que hemos podido comprobar durante la evaluación final del sistema es que *LM* tiende a mostrar un mejor comportamiento que *BM25* en general, haciéndose más visible a medida que el número de campos en el índice crece. El uso de este modelo de recuperación junto a las consultas booleanas, descritas en profundidad en el Anexo A.1.4, parece indicar que el uso de modelos de recuperación y consulta tradicionales basados en *BM25* o *BM25F* son menos óptimos para la estructuración del texto en campos que construye *Knowgly*.

Este comportamiento, de nuevo, se verificará al utilizar otros grafos de conocimiento, que podrían obligarnos a establecer diferentes estrategias de asignación de pesos o a ajustar los parámetros de los modelos de diferente forma, afectando así a sus resultados.