UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

# Descubrimiento y exploración en repositorios de datos

Informe de Proyecto de Grado presentado por

Nicolás Buero, Agustina Simoncelli, Raúl Maglione

En cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Lorena Etcheverry
Adriana Marotta

Montevideo, 3 de mayo de 2025

# Agradecimientos

Agradecemos profundamente a nuestras familias, cuyo esfuerzo y apoyo incondicional fueron imprescindibles para llegar hasta acá. Agradecemos también a nuestros amigos, tanto a los de siempre como a los que se sumaron en el camino, que nos hicieron sentir acompañados en cada paso. Finalmente, a todos los docentes que fueron parte de nuestra formación, y en particular a Lorena y Adriana por su apoyo y paciencia en el desarrollo de este proyecto.

# Resumen

El número de repositorios que contienen datos potencialmente útiles para futuros análisis ha crecido de manera significativa en los últimos años [19]. Si bien este incremento ofrece grandes oportunidades, también presenta desafíos significativos, especialmente en términos de tiempo y recursos que muchas veces se destinan a la preparación y recuperación de datos en lugar de centrarse en su análisis e interpretación.

El descubrimiento y la exploración de datos es un desafío, ya que requiere identificar varios conjuntos de datos interrelacionados dentro de enormes repositorios, donde los conjuntos de datos a menudo fueron producidos de manera independiente, y pueden incluir heterogeneidades como nombres de atributos o formatos de valores inconsistentes. Además, en muchos casos los metadatos existentes no son adecuados o suficientes, por lo que identificar qué conjuntos de datos pueden ser relevantes es una tarea costosa y que demanda esfuerzo.

Este trabajo tiene como objetivo realizar un relevamiento de las propuestas y herramientas disponibles para abordar los problemas de descubrimiento y exploración de datos, para aplicarlas sobre un repositorio de datos específico: el Catálogo de Datos Abiertos de Uruguay[1]. Este catálogo cuenta con aproximadamente 2500 conjuntos de datos publicados en distintos dominios, con formatos, niveles de especificación de metadatos y contenidos muy variados.

Para este propósito, se implementó un sistema que integra herramientas existentes enfocadas en la búsqueda, navegación y anotación de conjuntos de datos. Estas herramientas infieren tanto relaciones sintácticas entre columnas de distintos conjuntos de datos, así como relaciones semánticas. Además, utilizando los resultados de las herramientas y el contexto específico de cada conjunto de datos, se empleó un gran modelo de lenguaje (LLM por sus siglas en inglés) para generar automáticamente un archivo de *metadata* con descripciones enriquecidas de las tablas y sus columnas, proporcionando así una representación más clara y significativa de la información contenida en los datos.

Los resultados se integraron en un grafo de conocimiento que permite ejecutar consultas avanzadas, y con el uso de herramientas de visualización, permite navegar interactivamente por las relaciones sintácticas y semánticas entre los datos.

Finalmente, para los usuarios sin conocimientos técnicos en lenguajes de consulta, se implementó un enfoque basado en *Retrieval-Augmented Generation* (RAG), que permite realizar consultas en

[1]El Catálogo de Datos Abierto de Uruguay permite acceder a datos abiertos de organismos públicos, academia, organizaciones de sociedad civil y empresas privadas.

lenguaje natural de manera precisa y contextualizada.

Todas las herramientas utilizadas fueron evaluadas mediante experimentos diseñados para distintos escenarios de uso, ajustando hiperparámetros y calculando diversas métricas de evaluación para analizar su efectividad.

**Palabras clave:** Data lake, metadata, descubrimiento de datos, exploración de datos

# Índice general


**1. Introducción** **1**
- 1.1. Motivación . . . 2
- 1.2. Objetivos . . . 2
- 1.3. Organización del documento . . . 4

**2. Revisión de antecedentes** **7**
- 2.1. Data Lakes . . . 7
  - 2.1.1. Funciones dentro de un data lake . . . 8
- 2.2. Descubrimiento y Exploración de Conjuntos de Datos . . . 9
  - 2.2.1. Búsqueda de Conjuntos de Datos (Dataset Search) . . . 9
  - 2.2.2. Navegación de Datos (Data Navigation) . . . 10
  - 2.2.3. Anotación de Datos (Data Annotation) . . . 10
  - 2.2.4. Inferencia de Esquemas (Schema Inference) . . . 11
- 2.3. Resource Description Framework (RDF) . . . 11
- 2.4. Grandes Modelos de Lenguaje . . . 12
  - 2.4.1. Habilidades emergentes de los LLM . . . 12
  - 2.4.2. Retrieval-Augmented Generation . . . 12

**3. Trabajos relacionados** **15**
- 3.1. Descubrimiento de Conjuntos de Datos en Data Lakes (D3L) . . . 15
- 3.2. Aurum . . . 16
- 3.3. TableMiner + . . . 17
  - 3.3.1. Detección de la columna sujeto . . . 18
  - 3.3.2. Fase de aprendizaje . . . 19
  - 3.3.3. Fase de actualización . . . 20
  - 3.3.4. Consultas a la base de conocimiento . . . 21
  - 3.3.5. Selección de conceptos para anotar una columna . . . 23
- 3.4. Starmie . . . 24
  - 3.4.1. Etapa offline . . . 24
  - 3.4.2. Etapa online . . . 24
- 3.5. Solución de base utilizada . . . 25

**4. Análisis del caso de estudio** **27**
4.1. Catálogo Nacional de Datos Abiertos del Uruguay . . . 27
4.2. Extracción y Procesamiento de Datos . . . 29
4.2.1. Formatos seleccionados . . . 29
4.2.2. Descarga de Datos . . . 30
4.3. Limpieza de datos . . . 33

**5. Diseño e implementación de la herramienta propuesta** **35**
5.1. Pipeline . . . 36
5.2. Análisis de la solución de base . . . 38
5.2.1. Otras consideraciones . . . 39
5.3. Mejoras de la solución de base . . . 40
5.3.1. TableMiner+ . . . 40
5.3.2. Manejo de valores faltantes en celdas . . . 47
5.4. Generación de metadatos . . . 48
5.4.1. LLM utilizado . . . 48
5.4.2. Diseño del procedimiento . . . 48
5.5. Salidas del sistema . . . 50
5.5.1. Construcción de grafo RDF . . . 50
5.5.2. Consultas en lenguaje natural sobre los datos . . . 51
5.6. Mantenimiento del sistema . . . 52
5.6.1. Ingesta . . . 52
5.6.2. Borrado . . . 53
5.6.3. Actualización . . . 53

**6. Experimentación** **55**
6.1. Métricas de evaluación . . . 55
6.1.1. Precisión . . . 55
6.1.2. Recall . . . 55
6.1.3. F1 . . . 56
6.2. Entorno de ejecución . . . 56
6.2.1. Datos utilizados . . . 56
6.3. TableMiner+ . . . 57
6.3.1. Evaluación utilizando tablas de SemTab . . . 57
6.3.2. Evaluación utilizando tablas del Catálogo de Datos Abiertos de Gobierno . . . 62
6.4. Generación de metadatos . . . 65
6.4.1. Métricas de Evaluación . . . 66
6.4.2. Variantes de Ejecución . . . 66
6.4.3. Resultados de la evaluación . . . 67
6.5. RAG . . . 68
6.6. D3L y Aurum . . . 68
6.6.1. Diseño del experimento . . . 69
6.6.2. Resultados obtenidos . . . 69
6.7. Tiempo de construcción de índices LSH . . . 71

**7. Conclusiones y Trabajo Futuro** **73**
7.1. Conclusiones . . . 73
7.2. Trabajo a Futuro . . . 74
7.2.1. Análisis e incorporación de Starmie . . . 74
7.2.2. Vocabulario para el grafo RDF . . . 74
7.2.3. Copia del KG en ambiente local . . . 74
7.2.4. Similitud semántica por ancestros y descendientes . . . 74
7.2.5. Soporte para múltiples formatos de datos . . . 74
7.2.6. Entorno con mayores capacidades . . . 75
7.2.7. Interfaz visual para evaluación humana . . . 75
7.2.8. Implementación del mantenimiento del sistema . . . 75
7.2.9. Relaciones semánticas entre columnas de una misma tabla . . . 75

**A. Definiciones** **81**
A.1. Hashing Sensible a la Localidad (LSH) . . . 81
A.1.1. MinHash . . . 81
A.1.2. Proyecciones aleatorias . . . 82
A.2. TF-IDF . . . 82
A.3. Base de conocimiento . . . 83
A.3.1. Wikidata . . . 83

**B. Ejemplos de datos del Catálogo** **85**
B.1. Archivo de datos . . . 85
B.2. Archivo de metadatos . . . 85

**C. Prompts para el LLM** **87**
C.1. Generación de descripción . . . 87
C.2. Generación de descripción usando archivo de metadatos . . . 88
C.3. Generación de archivo de metadatos . . . 89
C.4. Generación de concepto candidato para columna . . . 90
C.4.1. Ejemplo 1 . . . 90
C.4.2. Ejemplo 2 . . . 91
C.5. Respuesta a consulta en lenguaje natural . . . 92

**D. Consultas SPARQL a la base de conocimiento** **95**
D.1. Buscar entidad dado el contenido de la celda . . . 95
D.2. Buscar triplas asociadas a una entidad . . . 95
D.3. Buscar conceptos asociados a una entidad . . . 96
D.4. Buscar descripción corta de una entidad . . . 96

**E. Guía para la ejecución de la herramienta implementada** **97**

# Capítulo 1

# Introducción

El número de conjuntos de datos que son utilizados por las organizaciones ha aumentado significativamente [6]. Aunque resulta sencillo acumular datos con un alto potencial para el análisis, la falta de una estructura clara dificulta identificar los conjuntos de datos más útiles para una tarea específica. Como consecuencia, los analistas suelen invertir más tiempo buscando datos relevantes para responder a sus preguntas que realizando el análisis propiamente dicho.

A estos repositorios de datos que carecen de un orden o estructura, se los conoce como *data lakes*. Aunque aún no existe un consenso claro sobre la definición exacta de un *data lake*, en este trabajo se adopta la definición que lo describe como un repositorio conformado por conjuntos de datos de los cuales solo se conocen metadatos básicos, como los nombres de los atributos (en el caso de datos tabulares) y, en algunos casos, sus tipos de datos [6]. Además, los formatos de almacenamiento pueden ser diversos, entre los cuales se encuentran: CSV, JSON, hoja de cálculo, tablas, texto, etc.

Los *data lakes* ofrecen la ventaja de almacenar grandes volúmenes de datos heterogéneos en sistemas de bajo costo. Sin embargo, presentan el riesgo de transformarse en pantanos de datos (*data swamps*), en los que los datos se vuelven difíciles de localizar, interpretar y utilizar de manera efectiva. Esto provoca una pérdida de valor analítico debido a la falta de estructura, control y gobernanza de los datos [14].

En este contexto, el descubrimiento y la exploración de datos se convierten en tareas esenciales para optimizar el proceso de análisis. La capacidad de identificar de manera eficiente los conjuntos de datos más relevantes y comprender las relaciones entre ellos, puede reducir significativamente los costos en tiempo y recursos asociados con la preparación de datos.

En este trabajo, se toma como caso de estudio el Catálogo de Datos Abiertos del Gobierno de Uruguay, considerado un ejemplo representativo de *data lake*. El objetivo es seleccionar, adaptar, complementar y aplicar herramientas diseñadas para resolver las tareas de descubrimiento y exploración de datos en este repositorio.

En las siguientes secciones de este capítulo se presenta la motivación del trabajo y se definen los objetivos del proyecto.

## 1.1. Motivación

Para motivar este trabajo se considera como ejemplo el caso de un analista de datos que desea investigar las desigualdades de género en los altos cargos de las instituciones estatales del Uruguay. Su objetivo es evaluar si existen diferencias significativas en la representación de hombres y mujeres en posiciones de liderazgo y cómo estas disparidades podrían estar relacionadas con otros factores, como el nivel educativo, el acceso a oportunidades laborales o las brechas salariales. Para ello, necesita combinar información de diferentes fuentes y temas como distribución de cargos, nivel educativo, ingresos económicos según sexo, entre otros.

El analista recurre al Catálogo de Datos Abiertos de Gobierno y tiene posibilidad de buscar por palabras claves o utilizando filtros como etiquetas, organizaciones o categorías. Sin embargo, enfrenta varios desafíos.

Uno de los problemas identificados es en la búsqueda por etiquetas, ya que esta depende de qué tan bien estén etiquetados los datos. Por ejemplo, al realizar la búsqueda por la palabra clave "Altos Cargos", como en la figura 1.1, se observa que entre los resultados se incluye "Cantidad de altos cargos del PIT CNT por sexo". Si bien el título del conjunto de datos aclara que los datos están distribuidos por sexo, no presenta una etiqueta relacionada al sexo como "Género". En caso de iniciar la búsqueda por esta etiqueta (lo que resulta natural por el tema de análisis), no se hubiese encontrado este *dataset*.

Otro inconveniente al buscar datos para un análisis que involucra diversos temas (en este caso educación, trabajo, género), es que se necesita realizar más de una búsqueda para abordar todos ellos. Por ejemplo, si se busca por etiquetas "Educación" y "Trabajo", solo se devuelven resultados que tengan ambas etiquetas, entonces para abordar la unión de ambos temas se deberían realizar dos búsquedas.

Otra característica deseable, sería que la información adicional del *dataset* (título, descripción, etiquetas, etc) sea suficiente para no necesitar descargar los recursos manualmente para saber si son útiles. Sin embargo, en el catálogo existen datos que presentan descripciones muy cortas o incluso vacías. En la figura 1.2 se puede ver dos ejemplos de descripciones vacías.

Finalmente, en caso de encontrar un *dataset* relevante para la investigación, no existe un mecanismo para encontrar otros *datasets* similares. Es decir, no existen relaciones entre los *datasets* que permitan navegación de datos.

Como resultado, el analista dedica más tiempo a buscar, limpiar y explorar datos que a analizarlos y generar información útil.

## 1.2. Objetivos

El objetivo de este trabajo es diseñar e implementar un sistema que facilite el análisis de datos abiertos mediante la integración de relaciones semánticas y sintácticas, optimizando la manera en que los analistas descubren, exploran y consultan conjuntos de datos. Este sistema estará orientado tanto a usuarios analistas de datos, con conocimientos técnicos (por ejemplo lenguaje de consultas) como a un investigador de otras disciplinas, que necesita obtener información compleja de múltiples


Figura 1.1: Interfaz de búsqueda del Catálogo de datos Abiertos de Uruguay por palabras clave

fuentes de datos a través de consultas en lenguaje natural. Para esto, se plantean los siguientes objetivos específicos:

- Realizar un relevamiento de las propuestas y herramientas disponibles para abordar los problemas de descubrimiento y exploración de datos en *data lakes*. Este análisis se enfocará en técnicas que permitan identificar relaciones sintácticas y semánticas entre datos heterogéneos.
- Explorar los datos del catálogo y obtener medidas que permitan evaluar la calidad actual de los datos y metadatos.
- Implementar un sistema que integre herramientas existentes para facilitar el descubrimiento y la exploración de datos en el Catálogo de Datos Abiertos de Uruguay. Esto incluirá la detección automática de relaciones entre columnas de diferentes conjuntos de datos, tanto desde una perspectiva sintáctica como semántica. Estos resultados serán integrados en una herramienta de exploración interactiva que permita descubrir conjuntos de datos relevantes a partir de un requerimiento específico. La herramienta facilitará la navegación a través de relaciones inferidas entre los datos, permitiendo expandir las conexiones y explorar conjuntos relacionados.


Figura 1.2: Interfaz de búsqueda del Catálogo de datos Abiertos de Uruguay por organización y categoria

- Evaluar el desempeño y los resultados de estas herramientas en el contexto del Catálogo de Datos Abiertos, analizando su efectividad en la mejora del proceso de descubrimiento y exploración de datos.

## 1.3. Organización del documento

Este documento se organiza en seis capítulos, abordando desde los fundamentos teóricos hasta la implementación y evaluación de las soluciones propuestas.

El capítulo 1 introduce el contexto y la motivación del estudio, definiendo el problema de investigación, la motivación y los objetivos específicos del trabajo.

El capítulo 2 desarrolla una revisión de los antecedentes teóricos, incluyendo definiciones de conceptos fundamentales para comprender el trabajo realizado.

El capítulo 3 analiza el estado del arte, explorando investigaciones previas y herramientas existentes para resolver el problema. Al final del capítulo se detalla una propuesta que integra varias herramientas en un *pipeline*, que fue la solución de base del presente trabajo.

En el Capítulo 4 se describe el Catálogo de Datos Abiertos de Uruguay y se detalla la preparación de los datos y la evaluación de calidad de datos realizada.

El Capítulo 5 detalla el diseño e implementación de la herramienta propuesta. Esta incluye las mejoras realizadas a la solución de base y los nuevos aportes introducidos.

El Capítulo 6 presenta la evaluación de las herramientas, incluyendo los conjuntos de datos seleccionados, el diseño de la experimentación, las métricas utilizadas, y los resultados obtenidos.

Finalmente, el Capítulo 7 expone las conclusiones del estudio, destacando los principales hallazgos, limitaciones y aportes del trabajo, junto con una discusión sobre posibles líneas de investigación futura.

Adicionalmente, el documento incluye un Anexo con información complementaria.

# Capítulo 2

# Revisión de antecedentes

En este capítulo se presentan conceptos necesarios para comprender el trabajo realizado.

## 2.1. Data Lakes

Con el aumento en el volumen y la variedad los datos, cada vez son más difíciles las tareas de recolección, almacenamiento y procesamiento [14].

En este contexto, los enfoques *schema-on-write*[1] pueden tornarse un poco ineficientes. Los *data lakes* surgen para evitar o posponer los costosos procesos de preprocesamiento, como la extracción, transformación y limpieza, que eran obligatorios en los *data warehouses*[2].
En escencia, un *data lake* es un sistema de almacenamiento de datos escalable y flexible que ingiere los datos en su formato original desde diversas fuentes. Estos permiten mantenimiento, procesamiento de consultas y análisis de datos de forma *schema-on-read*[3], con la ayuda de metadatos. Los datos abiertos de gobierno son un ejemplo de *data lake* [6].

Las características deseables (pero no usuales) de un *data lake* son: metadatos que garantizan la calidad de los datos, políticas y herramientas de gobernanza de datos, accesibilidad para diversos tipos de usuarios, integración de cualquier tipo de datos, una organización lógica y física y escalabilidad en términos de almacenamiento y procesamiento [24].

En particular, la gobernanza de datos abarca todas las medidas adoptadas para que los datos sean seguros, privados, precisos y accesibles. Incluye todas las acciones que deben realizar los usuarios, los procedimientos que deben seguir y la tecnología que utilizan durante todo el ciclo de vida de los datos [1].

[1] Previo a la carga de los datos se debe definir el esquema que estos deben cumplir.
[2] Colección de datos que es variable en el tiempo, orientada a un tema, integrada y no volátil. Siguen un enfoque schema-on-write.
[3] La definición de esquemas, integración de datos o indexado de datos se realiza al momento de acceder a los mismos.

### 2.1.1. Funciones dentro de un data lake

En la figura 2.1 se presenta una posible arquitectura de *data lake*. Esta se basa en las funciones necesarias para almacenar y consultar datos dentro de un *data lake*: ingesta, mantenimiento y exploración [14].
Se utiliza esta arquitectura a modo de ejemplo para visualizar estas funciones.

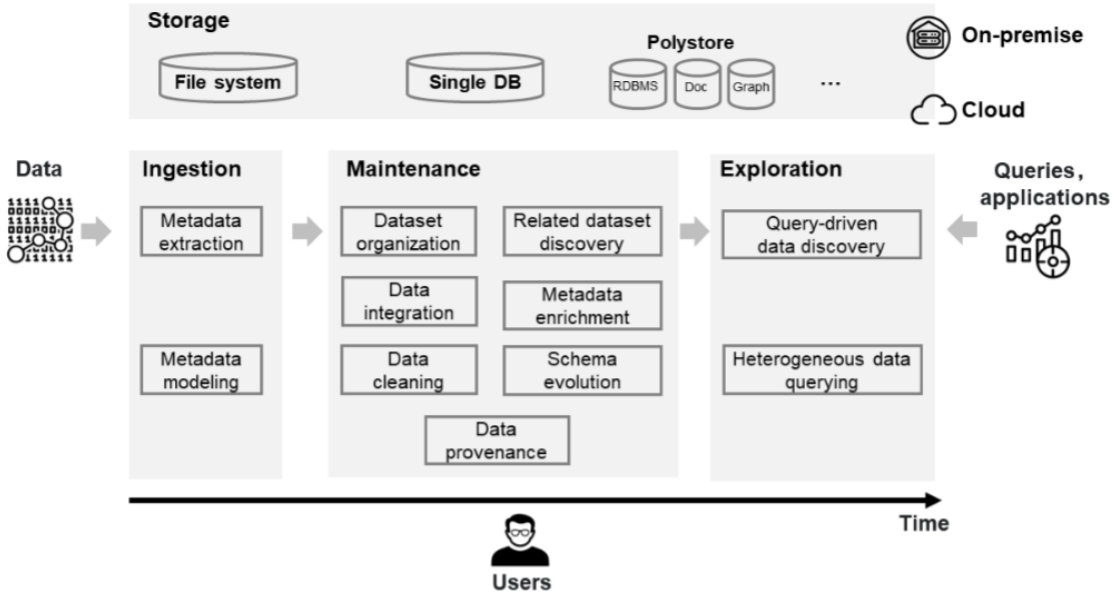


Figura 2.1: Arquitectura de data lake [14].

Un aspecto importante de la arquitectura de un *data lake* es el almacenamiento. Las soluciones para el almacenamiento pueden ser archivos, bases de datos, almacenamiento en la nube, o una combinación de estos.

La primer etapa que atraviesa un conjunto de datos al ingresar a un *data lake* es la ingesta de los datos crudos. Sin metadatos, como la estructura y semántica de los datos es desconocida, existe el riesgo de que el *data lake* se convierta en un *data swamp*, donde los datos se vuelven inaccesibles. Para esto, durante la ingesta o inmediatamente después, es necesario contar con mecanismos de extracción y modelado de metadatos.

Hasta este punto, luego de la ingesta solo se tiene una colección de datos sin relación alguna y del cual se tiene información limitada. Para darle utilidad a los datos, es necesario procesar y mantener los datos crudos. Para esto, en la etapa de mantenimiento se busca enriquecer la metadata, descubrir relaciones entre los datos, e integrar, transformar y limpiar los datos si es necesario.

Finalmente, es importante poder extraer información útil del *data lake*. En general se resuelve el problema de consulta de datos de dos maneras posibles: se explora el *data lake* a partir de relaciones

entre los *datasets* o se provee una interfaz de consulta unificada para datos heterogéneos.

## 2.2. Descubrimiento y Exploración de Conjuntos de Datos

Las tareas de descubrimiento y exploración de datos son esenciales para permitir un uso eficiente, preciso y significativo de los mismos [20]. El descubrimiento de datos es el proceso de identificar conjuntos de datos que pueden satisfacer una necesidad de información. Esto puede lograrse, por ejemplo, mediante una búsqueda directa, navegando desde *datasets* relacionados o explorando otros con una anotación específica. Por otro lado, la exploración de datos es el proceso de comprender sus propiedades y las relaciones entre ellos. Esto puede realizarse, por ejemplo, explorando las relaciones de un *dataset* determinado, visualizando anotaciones compartidas a nivel de *dataset* o atributos del mismo, o analizando relaciones que se comparten entre varios *datasets* dentro de un esquema inferido [19].

Se han desarrollado modelos enfocados en el uso de metadatos dentro de los catálogos de datos. Estos modelos implican la introducción de políticas de gobernanza de datos para garantizar que los datos sean seguros, privados, precisos, disponibles y utilizables así como políticas de registro y uso de los mismos. Sin embargo, asegurar el cumplimiento de estas políticas de manera manual resulta impráctico debido al gran volumen de datos que estos repositorios suelen manejar.

En respuesta a estas limitaciones, se han desarrollado técnicas automatizadas que emplean ontologías, buscan relaciones dentro de los conjuntos de datos, identifican dominios y analizan patrones recurrentes.
Estas técnicas suelen dividir el proceso de análisis del conjunto de datos en varias etapas como: búsqueda de conjuntos de datos, navegación de datos, anotación de datos, e inferencia de esquemas. A continuación, se describen estas etapas según las definiciones de los trabajos [19][20].

### 2.2.1. Búsqueda de Conjuntos de Datos (Dataset Search)

La búsqueda de conjuntos de datos, es el proceso de identificar *datasets* que, individualmente o en conjunto, satisfagan una necesidad de información. Una solicitud puede tomar diferentes formas, como palabras clave, un conjunto de datos o una consulta. Los conjuntos de datos se clasifican según el grado en el que se predice que satisfacen los requisitos expresados en la búsqueda.

A continuación se mencionan tres tipos de búsqueda de conjuntos de datos.

- **Búsqueda basada en palabras claves:** Esta tarea devuelve una lista de *datasets* relevantes con respecto a una o varias palabras clave. Esto permite a los usuarios localizar datos con un conocimiento mínimo de la estructura y las relaciones del conjunto de datos.
  La búsqueda por palabras clave puede aplicarse al encabezado, cuerpo o anotaciones de un conjunto de datos

- **Búsqueda basada en datasets:** Dado un *dataset*, tal vez derivado de otras tareas de búsqueda o navegación, busca otros con valores similares en el encabezado o cuerpo, por ejemplo recuperar conjuntos de datos individuales que sean compatibles en unión con un conjunto de datos dado.

- **Búsqueda basada en consultas:** Se utiliza un lenguaje de consultas sobre un modelo que representa las columnas de un repositorio y sus relaciones.

### 2.2.2. Navegación de Datos (Data Navigation)

La navegación de datos, es el proceso de explorar una colección de conjuntos de datos siguiendo las relaciones entre ellos.
Una vez identificado un *dataset*, por ejemplo, como resultado de un proceso de búsqueda, a menudo es útil poder explorar *datasets* relacionados que puedan proporcionar información o contexto adicional. Las relaciones utilizadas en la navegación suelen ser entre los atributos de los *datasets* y se suelen seguir distintas categorías de relaciones que resultan útiles. Las principales técnicas para identificar y descubrir estas relaciones son:

- **Descubrimiento de rutas de unión:** Identifica relaciones entre conjuntos de datos que pueden sustentar uniones (*join paths*).
- **Similitud semántica:** Identifica relaciones de similitud entre conjuntos de datos que pueden no admitir una unión, por ejemplo debido a representaciones disjuntas o inconsistentes. Por lo tanto, la similitud semántica resalta dónde se representan conceptos relacionados en un repositorio.
- **Descubrimiento de dominios:** Identifica atributos de *datasets* que comparten un dominio, donde un dominio es una colección de valores que representan un concepto de aplicación.

### 2.2.3. Anotación de Datos (Data Annotation)

La anotación de datos, se centra en enriquecer la metadata y la semántica de los datos mediante la asociación de elementos de datos con términos de un vocabulario o conceptos de una ontología.

En un repositorio grande, es probable que existan diferentes convenciones de nombres, por ejemplo, porque los datos fueron producidos originalmente por diferentes editores. El objetivo de la anotación de datos es obtener una comprensión más amplia y precisa de los datos. Esto permite, por ejemplo, discernir entre entidades con el mismo nombre pero en diferentes contextos.

Los *datasets* pueden anotarse en diferentes niveles de detalle:

- **Anotación de entidades:** Se aplica a nivel de instancia para precisar la interpretación de una palabra. Por ejemplo, para aclarar si la palabra "Uruguay" refiere al país o a la calle.
- **Anotación de tipos semánticos:** Asocia un *dataset* completo con una anotación. Por ejemplo, el tipo semántico de una tabla podría inferirse de las anotaciones de entidad de su atributo sujeto, o de los encabezados de columna.
- **Anotación de propiedades:** Proporciona anotaciones a atributos individuales, potencialmente con tipos literales o tipos semánticos.

Si bien el uso de ontologías externas aporta nueva evidencia a las tareas de descubrimiento y exploración de conjuntos de datos, la anotación efectiva depende de la cobertura de la ontología, y existe el riesgo de que la ontología disponible no refleje los aspectos del dominio relevantes para la tarea en cuestión.

### 2.2.4. Inferencia de Esquemas (Schema Inference)

La inferencia de esquemas, busca generar un esquema global que pueda representar los datos capturados por múltiples *datasets*, basándose en la identificación de conjuntos de datos que comparten elementos estructurales.

En un gran repositorio, a menudo existen múltiples *datasets* que describen un solo concepto. Por lo tanto, esta tarea sobre una colección de *datasets* infiere un esquema que representa los datos en el repositorio, pero que típicamente es mucho más pequeño que el esquema original.

El esquema inferido puede tener diferentes propósitos, entre los que se mencionan:

- **Consultas:** El objetivo es inferir un esquema que describa con precisión los datos subyacentes, para que puedan ser consultados.
- **Documentación:** El objetivo es identificar características recurrentes en el repositorio subyacente, como muchos conjuntos de datos que representan datos de empresas, que pueden agruparse como un solo tipo en un esquema inferido.
- **Resumen:** El objetivo es identificar las características más importantes en el repositorio subyacente y presentarlas como representativas del repositorio en su conjunto.

Aunque las cuatro etapas detalladas anteriormente parecen distintas a primera vista, existen superposiciones y sinergias entre ellas. Por ejemplo, la búsqueda de conjuntos de datos puede beneficiarse de las relaciones entre *datasets*, permitiendo identificar tablas que, al unirse, podrían satisfacer una solicitud de búsqueda. Asimismo, la inferencia de esquemas puede operar sobre los resultados de una búsqueda, en lugar de intentar inferir el esquema completo de un repositorio, lo que sería potencialmente voluminoso. Además, los resultados de la anotación de *datasets* podrían informar búsquedas o identificar qué *datasets* pertenecen al mismo tipo en el contexto de la inferencia de esquemas.

## 2.3. Resource Description Framework (RDF)

RDF [28] es un lenguaje para representar información en la Web. Este lenguaje permite modelar información compleja mediante grafos, facilita la integración de datos provenientes de diferentes dominios y asegura su reutilización gracias al uso de estándares globales reconocidos.

Un grafo RDF está compuesto por nodos y arcos dirigidos etiquetados que enlazan pares de nodos. Esto se representa como un conjunto de triplas RDF, donde cada tripla contiene un nodo sujeto, un predicado y un nodo objeto. Esto permite una representación visual de la información, donde los nodos corresponden a los sujetos y objetos y las aristas a los predicados que los conectan.

RDF no solo es flexible y escalable, sino que también admite la construcción de estructuras de conocimiento avanzado mediante herramientas adicionales como ontologías.

## 2.4. Grandes Modelos de Lenguaje

Los LLMs se refieren a modelos de lenguaje basados en *Transformers* [27] que contienen cientos de miles de millones (o más) de parámetros, entrenados con grandes volúmenes de datos textuales. Estos modelos tienen una gran capacidad para reconocer patrones en el lenguaje natural y resolver tareas complejas mediante la generación de texto [31].

### 2.4.1. Habilidades emergentes de los LLM

Las habilidades emergentes se definen formalmente como *habilidades que no están presentes en modelos pequeños, pero que surgen en modelos grandes* [31]. Esta es una de las características que diferencian a los LLMs de los modelos de lenguaje preentrenados anteriores (PLMs). Dos de estas habilidades son aprendizaje en contexto y seguimiento de instrucciones:

#### Aprendizaje en contexto (In-Context Learning, ICL)

Si al modelo se le proporciona una instrucción en lenguaje natural y/o varias demostraciones de una tarea, puede generar la salida esperada para los ejemplos de prueba simplemente completando la secuencia de palabras del texto de entrada, sin necesidad de entrenamiento adicional ni actualización de gradientes.

#### Seguimiento de instrucciones (Instruction Following)

Mediante el *fine-tuning*[4] con un conjunto de datos de múltiples tareas formateado como descripciones en lenguaje natural (*instruction tuning*), los LLMs han demostrado ser capaces de resolver tareas no vistas que también se presentan en forma de instrucciones. Gracias esta habilidad, los LLMs pueden seguir instrucciones de tareas nuevas sin requerir ejemplos explícitos, lo que mejora su capacidad de generalización.

### 2.4.2. Retrieval-Augmented Generation

Si bien los LLMs demuestran capacidades poderosas, aún enfrentan desafíos en aplicaciones prácticas, como alucinaciones, actualizaciones de conocimiento lentas y falta de transparencia en las respuestas [13]. Estos problemas se vuelven particularmente evidentes en tareas intensivas en conocimiento, donde es necesario acceder a una gran cantidad de información, como en preguntas de dominio abierto y en razonamiento basado en sentido común.

Un RAG, antes de responder preguntas o generar texto, primero recupera información relevante de un vasto corpus de documentos. Posteriormente, utiliza esta información recuperada para generar respuestas o texto, mejorando así la calidad de las predicciones. Esto permite a los desarrolladores evitar la necesidad de reentrenar un modelo grande para cada tarea específica. En su lugar, pueden adjuntar una base de conocimiento, proporcionando información adicional al modelo y aumentando la precisión de sus respuestas.

El RAG cuenta con dos etapas:

[4] Proceso de reentrenamiento de un modelo preentrenado para que se adapte a un conjunto de datos específico.

- Fase de Recuperación (Retrieval): Se utilizan modelos de codificación para recuperar documentos relevantes basados en la consulta.
- Fase de Generación (Generation): Usando el contexto recuperado como condición, el sistema genera texto.

Se ha demostrado que RAG mejora significativamente la precisión de las respuestas, reduce las alucinaciones del modelo, y se ha convertido en una de las tecnologías más utilizadas para mejorar los chatbots y hacer que los LLMs sean más aplicables en la práctica [13].

# Capítulo 3

# Trabajos relacionados

En este capítulo se presenta el estudio del estado del arte realizado de las propuestas y herramientas disponibles para abordar los problemas de descubrimiento y exploración de datos en *data lakes*. En particular, se analizaron los trabajos [6][11][30][10], que proponen herramientas para abordar las etapas de búsqueda, navegación, anotación e inferencia de esquemas respectivamente.

En la última sección del capítulo se presenta la solución de base utilizada, que surge de un repositorio [21][1] basado en los trabajos [19][20]. Este proporciona la implementación de un proceso secuencial de búsqueda, navegación, anotación e inferencia de esquemas sobre repositorios de datos.

## 3.1. Descubrimiento de Conjuntos de Datos en Data Lakes (D3L)

D3L [6] es una herramienta diseñada para facilitar la identificación y recuperación de *datasets* dentro de *data lakes*. Es especialmente útil en entornos con una cantidad limitada de metadatos y donde es necesario identificar relaciones entre atributos de diferentes *datasets* que no poseen un esquema específico o el esquema no está incluido en los metadatos.

El objetivo inicial es identificar atributos relacionados. Estrictamente hablando, esto solo puede lograrse si los atributos almacenan valores del mismo tipo de propiedad de una misma entidad del mundo real. Sin embargo, los *data lakes* suelen carecer de metadatos, lo que obliga a decidir si dos atributos de dos *datasets* están relacionados basándose únicamente en la evidencia que los propios *datasets* proporcionan.

D3L se basa en la similitud, ya que a partir de los nombres y valores de atributos en cada conjunto de datos del *data lake*, se extraen características que transmiten señales de similitud. Estas características se mapean en índices LSH (Ver anexo A.1), garantizando que la pertenencia compartida a un mismo contenedor sea indicativa de similitud según la función hash utilizada.
Para determinar la similitud entre atributos, D3L utiliza cinco tipos de evidencia:

[1] https://github.com/PierreWoL/EDBTDemo

- **Nombres (N):** Se transforman los nombres de atributos en q-gramas[2]. La similitud entre los nombres de atributos se calcula como la distancia de Jaccard (Ver anexo A.1.1) entre sus *qsets*.
- **Valores (V):** Los valores de los atributos se transforman en un conjunto de *tokens* informativos (*tset*), definidos mediante métricas similares a TF/IDF (Ver anexo A.2) de recuperación de información. La similitud entre los nombres de atributos se calcula como la distancia de Jaccard entre sus *tsets*.
- **Formatos (F):** Los valores de los atributos se representan mediante un conjunto de expresiones regulares (*rsets*), que describen su estructura predecible (correos electrónicos, fechas, URIs). La similitud entre formatos de valores de atributos se mide como la distancia de Jaccard entre sus *rsets*.
- **Word embeddings (E):** Para valores con contenido textual, se utiliza un modelo de incrustación de palabras[3] (*word embedding model, WEM*). Cada palabra recibe un vector de dimensión $p$, y los vectores se combinan en un vector representativo del atributo. La similitud entre atributos con contenido textual se mide como la distancia coseno (Ver anexo A.1.2) entre sus vectores.
- **Distribuciones de dominio (D):** Para valores con contenido numérico, se usa la estadística de Kolmogorov-Smirnov[4] (*KS*) para medir la similitud en las distribuciones de los valores de atributos. Atributos con valores numéricos son más similares cuanto menor sea el valor de KS entre sus distribuciones.

Dado que las relaciones N, V y F se basan en la similitud Jaccard, y la relación E se basa en la similitud coseno, se utiliza MinHash y proyecciones aleatorias (Ver anexos A.1.1 y A.1.2 respectivamente) para aproximar estas distancias de manera eficiente. La relación D no utiliza esta estrategia porque no existe un esquema de hash LSH que ofrezca beneficios equivalentes.

## 3.2. Aurum

Aurum [11] es un sistema de descubrimiento que aborda la navegación de datos. Esta herramienta permite encontrar datos relevantes de manera flexible, a través de propiedades de los *datasets* o relaciones sintácticas entre ellos, como la similitud de contenido o esquemas, o la existencia de enlaces de clave primaria/clave foránea (PK/FK).
Esta herramienta permite construir, mantener y consultar un grafo de conocimiento empresarial (*EKG* por sus siglas en inglés) que representa las relaciones entre los conjuntos de datos.

El EKG es un hipergrafo[5] donde los nodos representan columnas de los *datasets*, las aristas representan relaciones entre dos nodos, y las hiperaristas conectan cualquier número de nodos que

[2]División de texto en secuencias de q elementos consecutivos.
[3]Técnicas que proyectan palabras en un espacio vectorial de alta dimensionalidad capturando relaciones semánticas y contextuales entre términos, reflejadas en la proximidad de sus vectores.
[4]Prueba estadística que se utiliza para comparar distribuciones de probabilidad.
[5]Generalización de un grafo, cuyas aristas se llaman hiperaristas, y pueden relacionar a cualquier cantidad de vértices.

estén relacionados jerárquicamente, como las columnas de la misma tabla o las tablas de una misma fuente. Además, cada arista tiene un peso que indica la fortaleza de la relación.
Las aristas conectan únicamente dos columnas para expresar relaciones binarias, como:

- **Similitud de contenido:** Los valores de las columnas son similares.
- **Similitud de esquema:** Los nombres de los atributos son similares.
- **Existencia de PK/FK:** Una columna es clave primaria y la otra clave foránea.

## 3.3. TableMiner +

TableMiner+ [30] es una herramienta que aborda los problemas de la interpretación semántica de tablas para la anotación de datos.
La Interpretación Semántica de Tablas aborda tres tareas fundamentales de anotación:

- Enlazar nombres de entidades en celdas con entidades de referencia (desambiguación).
- Anotar columnas con conceptos semánticos si contienen menciones de entidades (*NE-columns*), o propiedades de atributos si contienen literales de datos (*literal-columns*).
- Identificar las relaciones semánticas entre columnas de una misma tabla.

El funcionamiento de TableMiner+ se puede dividir en tres fases (Ver figura 3.1): la detección de la columna sujeto, la fase de aprendizaje y la fase de actualización.

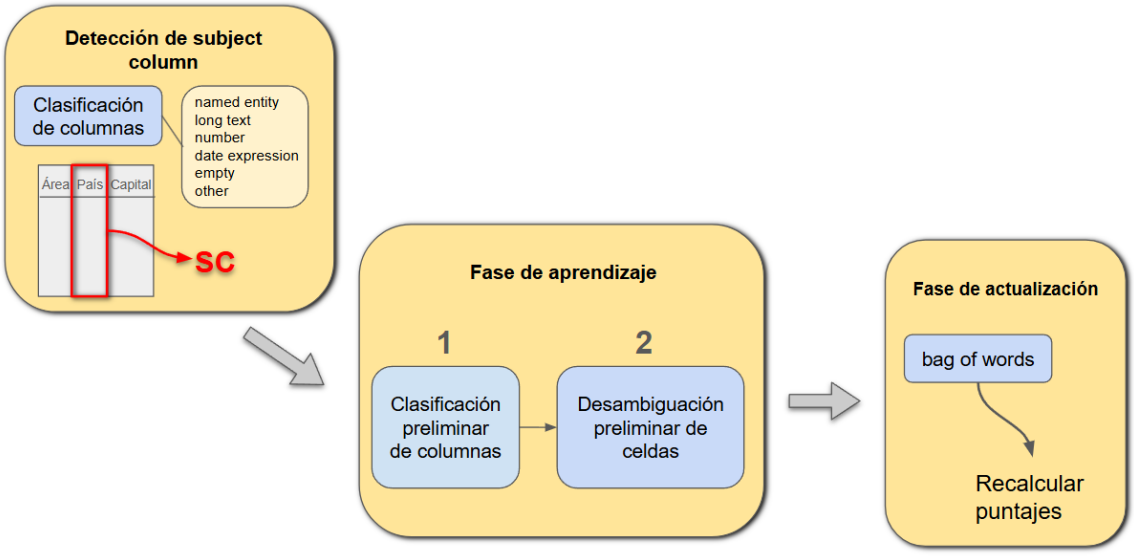


Figura 3.1: Funcionamiento de TableMiner+

### 3.3.1. Detección de la columna sujeto

La columna sujeto (*subject column*) de un conjunto de datos representa la entidad sobre la cual trata el conjunto de datos.
Esto aplica para tablas que describen una entidad específica donde hay una columna sujeto y las demás columnas son atributos del mismo, por ejemplo la tabla 3.1 donde la columna sujeto es "País" y las otras dos columnas se refieren a atributos de cada país. Sin embargo existen tablas que no tienen esta estructura, por ejemplo, la tabla 3.2, que muestra cierto valor dado un país, un sexo y un año, es decir, se requiere la tripla para identificar una fila.

| País | Población | Capital |
|---|---|---|
| Uruguay | 3.388.171 | Montevideo |
| Chile | 19.658.835 | Santiago de Chile |
| México | 128.455.567 | Ciudad de México |
| Colombia | 52.939.527 | Bogotá |
| Perú | 33.845.617 | Lima |

Tabla 3.1: Países con población y capital

| País | Sexo | Año | Valor |
|---|---|---|---|
| Brasil | Varones | 2011 | 55.7 |
| Brasil | Mujeres | 2011 | 44.3 |
| España | Varones | 2011 | 53.2 |
| Australia | Varones | 2011 | 59.3 |
| México | Varones | 2011 | 50.8 |

Tabla 3.2: Distribución porcentual de personas que residían en el exterior en el año 2006 según país de residencia por sexo.

Esta columna suele desempeñar un papel similar al de una clave primaria, especialmente al buscar relaciones entre conjuntos de datos o agrupar conjuntos de datos relacionados [19].

La detección de la *subject column*, comienza clasificando mediante expresiones regulares, las celdas de cada columna según uno de los siguientes tipos de datos: *empty*, *named_entity*, *number*, *date_expression*, *long_text*, *other*.

Posteriormente, para cada columna clasificada como *named-entity*, se calculan las siguientes características:

- **Proporción de celdas vacías:** Se calcula como el número de celdas vacías dividido por el número total de filas en la columna.
- **Proporción de celdas con contenido único:** Es la relación entre el número de celdas con contenido de texto único y el número total de filas.
- **Distancia desde la primera columna *named-entity*:** Cuenta cuántas columnas están separadas de la primera columna *named-entity* candidata desde la izquierda.

- **Detección de acrónimos o identificadores:** Se verifica con expresiones regulares el número de celdas que probablemente contengan acrónimos o identificadores. Si la columna se identifica como acrónimo o identificador, se desestima como *subject column*, ya que se prefiere que esta contenga nombres completos para mayor claridad.
- **Puntaje de coincidencia con el contexto:** Este puntaje evalúa la frecuencia con la que las palabras del encabezado de la columna aparecen en el contexto del encabezado.
- **Puntaje de búsqueda web:** Este puntaje utiliza evidencia de la web para predecir la probabilidad de que una columna sea la *subject column*, contribuyendo con más información de las filas individuales de la tabla.

Estas características se normalizan y se combinan para calcular un puntaje final de cada columna. La columna con mayor puntaje se elige como la *subject column* [30].

### 3.3.2. Fase de aprendizaje

Luego de detectar la *subject column*, comienza la fase de aprendizaje, cuyo objetivo es realizar una clasificación preliminar de columnas y desambiguación de celdas en cada columna clasificada como *named-entity* [30].

#### Clasificación preliminar de columnas

En esta etapa, TableMiner+ genera conceptos (tipos semánticos) candidatos para anotar cada columna y calcula puntajes de confianza para cada candidato, siendo los conceptos de mayor puntaje los ganadores.

En primer lugar (Ver figura 3.2), se realiza un proceso de desambiguación inicial (*cold-start disambiguation*) para generar conceptos candidatos para la columna a partir de anotaciones de entidad para cada celda. Este proceso se realiza solo para una muestra de los datos de la columna hasta alcanzar la convergencia (cuando de una iteración a otra no surgen cambios en los conceptos candidatos), y se selecciona el concepto ganador para anotar la columna. Estos conceptos generados pueden cambiar en fases posteriores.
En el ejemplo de la figura, se recorren las tres primeras celdas de la columna, tras lo cual el algoritmo converge. Para las celdas recorridas, se obtienen entidades asociadas al contenido de la celda y con el tipo semántico de estas entidades se encuentran dos conceptos candidatos para anotar la columna. Se elige uno de los conceptos como ganador y con este se anota la columna.

El tamaño y contenido de la muestra pueden afectar la precisión de la clasificación. Por esta razón, un proceso de ordenamiento de muestras precede la clasificación preliminar, reorganizando las filas de la tabla para optimizar la muestra y maximizar la precisión.

#### Desambiguación preliminar de celdas

En esta etapa (ver figura 3.3), la anotación de la columna creada en la clasificación preliminar se utiliza para desambiguar las anotaciones iniciales en las celdas que ya han sido procesadas, restringiendo el espacio de entidades candidatas (solo se conservan entidades asociadas con el concepto ganador de la etapa anterior). Las celdas que no han sido visitadas se visitan y se aplica la misma restricción del espacio de entidades candidatas.

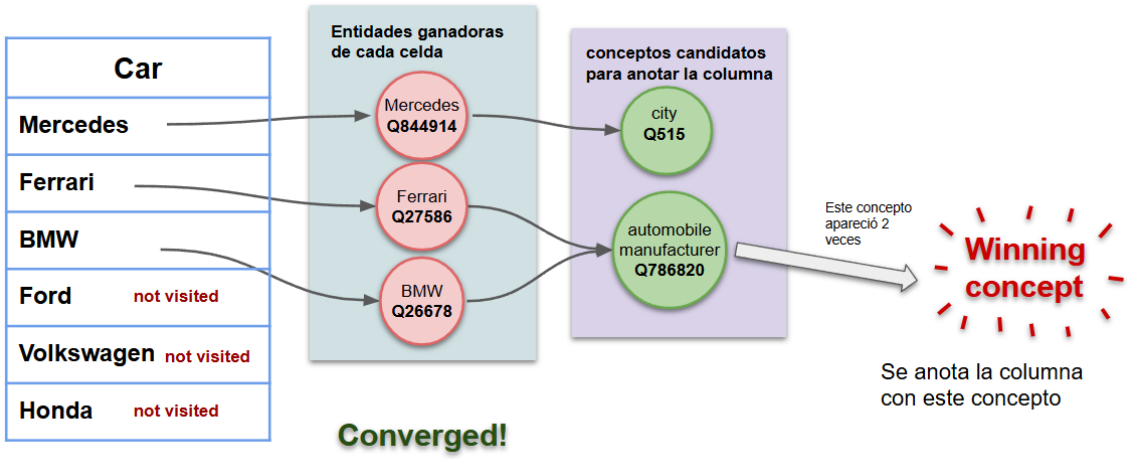


Figura 3.2: Clasificación preliminar de columnas

En el ejemplo de la figura, la celda "Mercedes" tiene dos entidades posibles (la referente a la ciudad y la referente a la marca de vehículo). En la desambiguación, la entidad correspondiente a la ciudad se descarta ya que no tiene el concepto ganador de la etapa anterior asociado, conservando solo la entidad correspondiente al vehículo. De esta forma se tiene en cuenta el contexto de la columna para anotar cada celda y esto permite la desambiguación.

La desambiguación de cada nueva celda genera un conjunto de conceptos de los cuales algunos pueden ser nuevos. Como esto modifica el conjunto de conceptos candidatos de la columna y sus puntajes, en algunos casos puede causar inconsistencias (por ejemplo que haya un nuevo concepto ganador), se requiere una fase de actualización para gestionar estas inconsistencias.
En el ejemplo de la figura surge el nuevo concepto "car brand".

### 3.3.3. Fase de actualización

La fase de actualización (Ver figura 3.4) refina las anotaciones iniciales realizadas en la fase de aprendizaje mediante un proceso iterativo, y utiliza tanto el contexto interno de las tablas como el contexto externo disponible.

En primer lugar, se construye una representación del dominio a partir de las anotaciones de entidades generadas en las celdas de las columnas *named-entity*. Esto se logra mediante consultas SPARQL[6] a la base de conocimientos (ver anexo A.3), específicamente para obtener las definiciones de todas las entidades ganadoras (*winning entities*) asociadas a las celdas. Con esta información,

[6]SPARQL es el estándar de consulta para RDF, que permite consultar patrones de grafos, sus conjunciones y disyunciones.

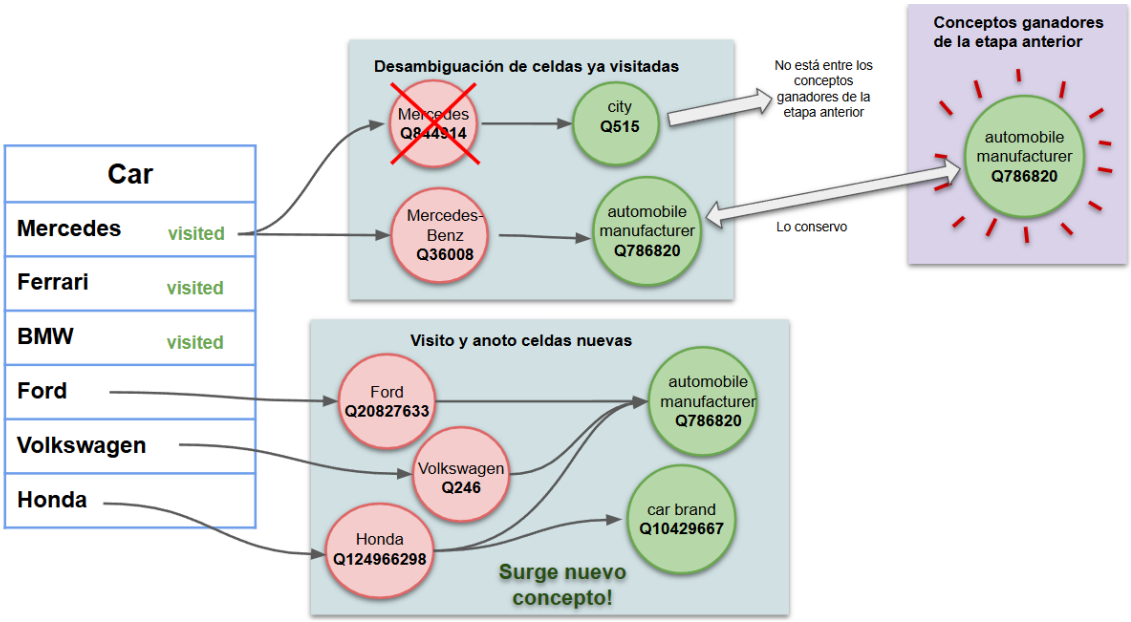


Figura 3.3: Desambiguación preliminar de celdas

se construye una *bag-of-words*[7].

A continuación, se recalculan los puntajes de cada concepto, incorporando un nuevo factor que mide la similitud entre el concepto y la *bag-of-words* de la columna.
Este recálculo no solo introduce el contexto proporcionado por la *bag-of-words*, sino que también responde a posibles variaciones en los puntajes de los conceptos tras el procesamiento completo de todas las celdas en la fase de desambiguación preliminar.

En segundo lugar, se revisan las anotaciones de columnas. Si se detectan cambios en las anotaciones de alguna columna, se selecciona un nuevo concepto ganador para la columna y se revisan las anotaciones de desambiguación en las celdas de la columna en cuestión.

Este proceso continúa hasta que no se produzcan más cambios en las anotaciones de columnas o celdas.

### 3.3.4. Consultas a la base de conocimiento

Para abordar las tres tareas principales de anotación (a nivel de celda, columna y propiedades), es necesario realizar consultas a una base de conocimiento. Como ejemplo, en esta sección se describen

[7]Método que representa el texto como un conjunto de palabras sin considerar su orden, permitiendo analizar la frecuencia de aparición de cada palabra.

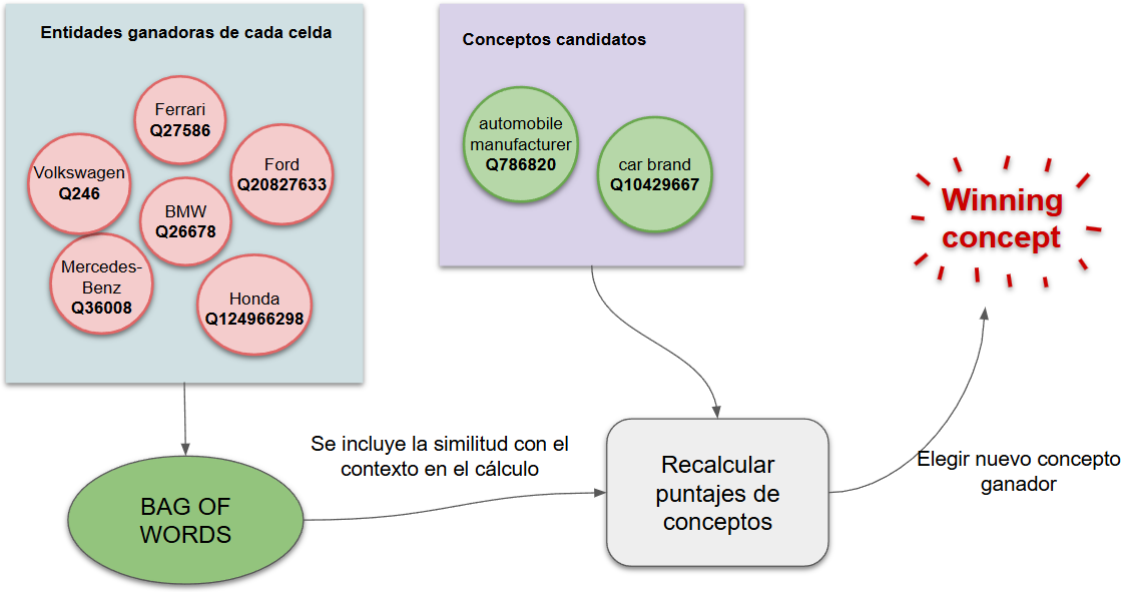


Figura 3.4: Fase de actualización de TableMiner+

las consultas utilizadas en la implementación de TableMiner+ del trabajo [21] para Wikidata (ver anexo A.3.1).

### Búsqueda de entidad dado el contenido de la celda

Para anotar las celdas con entidades de la base de conocimiento, se realiza una consulta que busca entidades cuya etiqueta coincida parcialmente con el contenido de la celda. Dado que se trata de una búsqueda de texto en la base de conocimiento, esta consulta suele ser más costosa en términos de tiempo de ejecución en comparación con otras consultas en las que ya se dispone de la URI[8] de la entidad.

En el anexo D.1 se muestra la consulta SPARQL utilizada para este propósito.

### Búsqueda de triplas asociadas a una entidad

Una vez obtenidas las entidades candidatas de una celda, se aplica un filtro adicional basado en la existencia de triplas asociadas. Las entidades que no tienen propiedades definidas en la base de conocimiento se descartan.

Para verificar esto, se consulta la estructura de triplas RDF de cada entidad recuperada. Si una entidad no tiene propiedades asociadas, se considera que no aporta información semántica útil para

[8]Identificador Uniforme de Recursos (*Uniform Resource Identifier*) es una cadena de caracteres que identifica un recurso.

la anotación de la celda y, por lo tanto, es descartada. Este criterio se basa en la idea de que una entidad sin propiedades documentadas es poco informativa y no contribuye significativamente a la interpretación de la tabla.

En el anexo D.2 se presenta la consulta SPARQL utilizada para recuperar las triplas asociadas a cada entidad.

#### Búsqueda de conceptos asociados a una entidad

Para cada entidad identificada en las celdas de una columna, se buscan los conceptos semánticos asociados, los cuales serán utilizados como candidatos para anotar la columna completa.

En el caso de Wikidata, estos conceptos pueden obtenerse a través de las propiedades wdt:P31 (instancia de) y wdt:P279 (subclase de), permitiendo identificar el tipo semántico de las entidades encontradas.

En el anexo D.3 se muestra la consulta SPARQL utilizada para recuperar estos conceptos asociados.

#### Búsqueda de descripción breve de una entidad

En la fase de actualización, se construye una representación del dominio a partir de las anotaciones de entidades generadas en las celdas de las columnas. Con esta información, se construye una *bag-of-words* con la cual se comparan los conceptos candidatos para evaluar su similitud contextual. Para construir esta representación del dominio, se recupera la descripción breve de cada entidad identificada en las celdas mediante una consulta SPARQL.

En el anexo D.4 se muestra la consulta SPARQL utilizada para obtener las descripciones de las entidades.

### 3.3.5. Selección de conceptos para anotar una columna

Una vez que se tienen entidades candidatas para todas las celdas de una columna, se procede a extraer los conceptos asociados a estas entidades, los cuales servirán como candidatos para la anotación de la columna completa. A partir de estos conceptos, se asignan puntajes de clasificación, seleccionando finalmente el concepto con la puntuación más alta como la anotación semántica definitiva de la columna.

En la siguiente sección se detallan los criterios y métodos utilizados en la implementación de TableMiner+ del trabajo [21] para calcular los puntajes que determinan el concepto final con el que se anota la columna.

#### Concepto ganador

La puntuación que determina el concepto ganador para anotar la columna se calcula a partir de dos métricas:

- **ConceptInstanceScore:** Dado un concepto, se suman los puntajes de todas las *winning-entities* que lo tienen asociado. Luego, este valor se divide por el total de filas anotadas. De esta manera, si varias celdas con alta confianza apuntan a entidades que forman parte de un mismo concepto, dicho concepto obtendrá un puntaje más elevado.

$$\text{ConceptInstanceScore}(c) \;=\; \sum_{\substack{e \in \text{winning\_entities} \\ \text{concept}(e) = c}} \frac{\text{CF}(e)}{\#annotated\_rows}.$$

- **ConceptContextScore:** Evalúa la coherencia entre el nombre del concepto y los contenidos textuales de la columna correspondiente (incluyendo el nombre de la columna). Para ello, se emplea el *Dice Coefficient*[9], que mide la similitud de conjuntos entre las palabras del concepto y el conjunto resultante de los textos de la columna:

$$\text{ConceptContextScore}(c, \text{col}) \;=\; \frac{2 \times |\text{bow}(c) \cap \text{bow}(\text{col})|}{|\text{bow}(c)| \;+\; |\text{bow}(\text{col})|}.$$

La integración de ambas métricas contribuye a refinar la clasificación preliminar de columnas, favoreciendo a aquellos conceptos que, por un lado, cuenten con instancias (*winning entities*) bien puntuadas y, por otro, se ajusten al contenido textual tanto de las celdas como del nombre de la columna.

## 3.4. Starmie

Starmie [10] es un sistema que aborda la inferencia de esquemas. Esta herramienta resuelve el problema de búsqueda de tablas unibles (*joineables*) en dos etapas: *offline* y *online*.

### 3.4.1. Etapa offline

Se entrena un modelo para codificar columnas de tablas de *data lakes* en *embeddings*. Luego se aplica el modelo entrenado a todas las tablas del *data lake* para obtener los *embeddings* de las columnas mediante inferencia del modelo. Finalmente, los vectores de *embeddings* se almacenan en índices vectoriales eficientes para su recuperación *online*.

### 3.4.2. Etapa online

Dada una tabla de consulta como entrada, se recupera un conjunto de tablas candidatas de los índices vectoriales buscando *embeddings* de columnas del *data lake* con alta similitud a nivel de columna con las columnas de entrada.
Luego se aplica un paso de verificación para clasificar las tablas candidatas según los puntajes de unionabilidad a nivel de tabla y obtener las k tablas con mayor puntaje.

[9] Medida de similitud entre dos cadenas que mide el porcentaje de la cantidad total de q-gramas de las dos cadenas que son idénticos.

## 3.5. Solución de base utilizada

El trabajo [21] provee una demostración con una implementación simplificada de las herramientas detalladas anteriormente. En particular, los autores implementan un sistema para el descubrimiento y la navegación de datos usando:

- D3L para búsqueda de *datasets*.
- Aurum para navegación de datos.
- TableMiner+ para anotación semántica.
- Starmie para inferencia de esquemas.

En la figura 3.5 se muestra la solución base utilizada. En primer lugar en D3L se construyen índices LSH, luego con los índices de nombres y valores de atributos se construye el grafo de Aurum, donde para cada columna se construye un nodo y se realiza una consulta utilizando ambos índices para obtener los "vecinos más cercanos", los filtra (similitud promedio entre los dos índices mayor a cierto umbral de similaridad) y construye una arista con los nodos seleccionados. En paralelo, se detecta el tipo de las columnas y las clasificadas como *named_entity* son anotadas con un concepto de Wikidata. Por separado, Starmie se encarga de buscar tablas que puedan ser unificadas para inferir un esquema global.

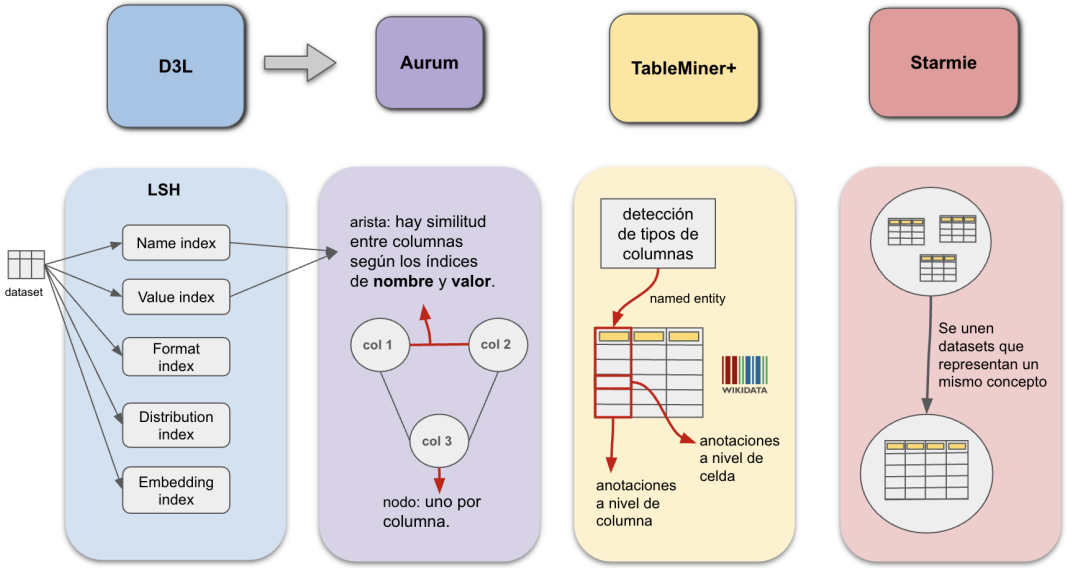


Figura 3.5: Sistema provisto en [21]

# Capítulo 4

# Análisis del caso de estudio

En este capítulo se analiza y se describe el Catálogo de Datos Abiertos de Uruguay y los recursos presentes en él. Se detalla un proceso de descarga y preprocesamiento de los datos, que fue implementado con el fin de seleccionar una porción significativa de datos del catálogo para adaptar y aplicar las herramientas al caso de estudio. En este proceso, se obtuvieron medidas que permitieron evaluar la calidad de los datos y metadatos.

## 4.1. Catálogo Nacional de Datos Abiertos del Uruguay

El Catálogo Nacional de Datos Abiertos [2] es una iniciativa que busca fomentar la transparencia y democratizar el acceso a la información pública. Esta herramienta permite acceder a datos abiertos de organismos públicos, academia, organizaciones de sociedad civil y empresas privadas [3]. Cualquier persona puede utilizar los datos publicados libremente para diversos fines, como la creación de aplicaciones, investigaciones académicas o análisis estadísticos.
Si bien el catálogo es gestionado por el equipo de Gobierno Abierto de AGESIC, la calidad de los datos recae en los publicadores.

El catálogo está desarrollado en base a CKAN (*Comprehensive Knowledge Archive Network*) como herramienta de catálogo. Es un sistema de gestión de datos, de código abierto diseñado para facilitar la creación de portales y centros de datos. Esta herramienta permite publicar, compartir y utilizar datos de manera sencilla a través de una API (*Application Programming Interface*), y actualmente ha sido adoptada por gobiernos de todo el mundo [8].

#### Recursos presentes en el catálogo

CKAN organiza los datos en paquetes (*packages*), que agrupan recursos individuales y sus metadatos. En la figura 4.1 se muestra un ejemplo de *package* desde la interfaz web del catálogo.

En primer lugar, cada *package* cuenta con un título y una descripción general (recuadros A y B en la figura). Estos proporcionan información sobre el contenido del *package* y facilita la comprensión de la naturaleza de los datos, así como el tipo de información que se puede esperar en los archivos disponibles.

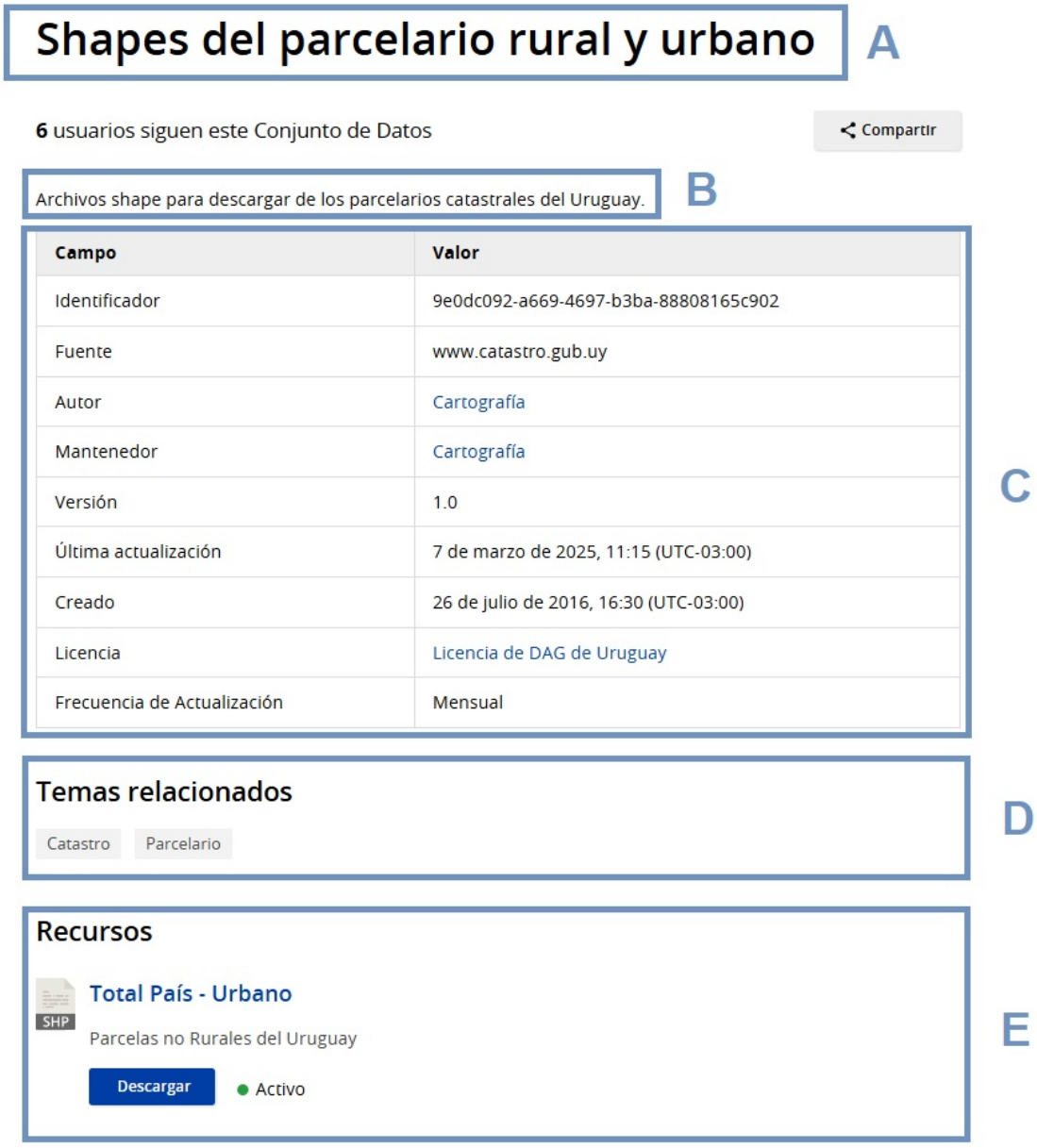


Figura 4.1: Interfaz de un package en el catálogo de datos

Además, en la sección de metadatos (recuadro C), se incluyen campos como: identificador, fuente, autor, mantenedor, versión, última actualización, fecha de creación, licencia y frecuencia de actualización. A continuación, se puede ver una sección de temas relacionados (recuadro D), con

etiquetas con las que está anotado el *package*.

Por último, cada *package* incluye la lista de recursos (recuadro E), que abarcan tanto los archivos de datos como sus metadatos en diversos formatos (XML, TXT, PDF, XLS, CSV, JSON, etc).
En la mayoría de los casos, los archivos de metadatos contienen información como el título del *dataset*, su descripción y una lista detallada de atributos, incluyendo su nombre, tipo de dato y una breve descripción. Se presenta un ejemplo de archivo de datos y metadatos en los anexos B.1 y B.2 respectivamente.

## 4.2. Extracción y Procesamiento de Datos

En esta sección, se describen las estrategias empleadas para seleccionar un subconjunto representativo del catálogo, las herramientas utilizadas para detectar y corregir inconsistencias, y las medidas obtenidas que reflejan la calidad de los datos.

El objetivo de esta etapa es garantizar que los datos extraídos sean adecuados para su posterior análisis y etiquetado automatizado, aplicando técnicas de preprocesamiento y limpieza de datos.

### 4.2.1. Formatos seleccionados

En el presente trabajo, se decidió trabajar exclusivamente con archivos de datos en formato CSV. En primer lugar, representan la mayoría de los *datasets* disponibles en el catálogo [4]. Por otro lado, la solución de base presentada en el capítulo anterior, solo acepta datos en este formato.
Para los recursos de metadatos, se optó por trabajar exclusivamente con archivos en formato JSON. Este formato es especialmente adecuado para la representación de metadatos debido a su capacidad para estructurar información jerárquica y compleja de manera clara y organizada.
Aunque el catálogo soporta otros formatos como XML, TXT, PDF y similares, ampliar el alcance para incluir todos ellos habría incrementado significativamente el alcance del proyecto, desviando así el objetivo principal.

Respecto a los conjuntos de datos seleccionados, en primera instancia, se decidió trabajar con recursos clasificados bajo la etiqueta “Transparencia Activa”. Esta decisión fue sugerida por un referente del catálogo de datos, quien destacó que los paquetes etiquetados de esta manera, suelen seguir formatos específicos y contar con metadatos estructurados.

A medida que avanzó el proyecto, se identificó que la mayoría de los conjuntos de datos en el catálogo consistían en tablas estáticas o tablas multidimensionales con valores de mediciones, con columnas mayormente numéricas, lo que limitaba la capacidad de análisis y la posibilidad de encontrar relaciones relevantes entre los datos. En consecuencia, se optó por ampliar el alcance del estudio descargando la totalidad de los conjuntos de datos del catálogo y realizando una selección manual de aquellos con mayor contenido significativo para análisis, priorizando los que presentaran más potencial para la identificación de relaciones sintácticas y semánticas.

### 4.2.2. Descarga de Datos

Para identificar y obtener *datasets* relevantes del Catálogo de Datos Abiertos de Uruguay, se empleó la búsqueda por etiquetas (*tags*) mediante la API de CKAN [7]. Estos *tags* aparecen en la interfaz web del catálogo tal y como se muestra en el recuadro D de la figura 4.1. Este tipo de consulta devuelve una lista de *packages* que contienen la etiqueta especificada, junto con información clave como la descripción del *package*, las etiquetas asociadas y los enlaces de descarga de los recursos (datos y metadatos).

Al explorar diversos conjuntos de datos del catálogo, se identifica que en cada *package* existen varios recursos que corresponden a tablas con los mismos datos para diferentes años. Procesar tablas casi con la misma información no aporta un valor significativo al análisis y, además, incrementa innecesariamente el tiempo de procesamiento. En base a esta observación, se decidió seleccionar para cada *package*, una única tabla representativa y la metadata correspondiente. Para ello se implementó un preprocesamiento de datos que permite seleccionar en el mejor de los casos, archivos que cumplen con los estándares de calidad esperados, es decir, sin errores y en los formatos requeridos (CSV para las tablas y JSON para la metadata).
El preprocesamiento realizado se muestra en la figura 4.2 y se detalla a continuación.

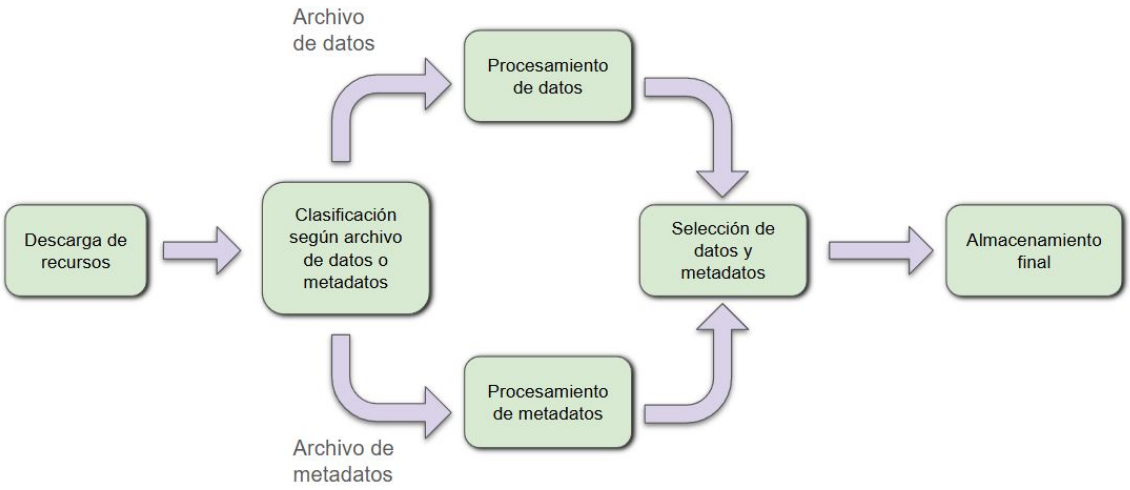


Figura 4.2: Secuencia de descarga y preprocesamiento de recursos

#### Descarga y clasificación de recursos

En esta etapa, se descargaron los recursos para cada *package*. Como CKAN no proporciona ningún mecanismo para saber si el recurso es de datos o de metadatos, se adoptaron los siguientes criterios basados en el nombre del recurso:

- Si este contiene en su nombre una palabra relacionada a metadatos como 'metadata', 'metadatos', 'descripción de los datos', 'datos descriptivos', o 'descripción', se define como potencialmente metadata y se almacena con el nombre *'potential_metadata_id'*, siendo *id* el identificador del recurso.

- En caso contrario, y si el formato es CSV, se define como un recurso de datos y se almacena con el nombre *'table_id'*, siendo *id* el identificador del recurso.

Para cada *package* inicialmente se descargan hasta 3 tablas, filtrando por tamaño de archivo inferior a 42MB.
Durante el proceso de descarga, se recopilaron medidas que muestran los problemas de calidad de datos encontrados. En particular, en esta etapa se registró la cantidad de *datasets* descargados exitosamente, así como aquellos que presentaron errores durante la descarga.

Además, se genera un archivo JSON llamado *additional info*, que almacena información contextual del paquete como título, descripción del *package*, organización que publica, fechas de creación y recursos. Esta información es extraída directamente del catálogo, siempre y cuando los publicadores hayan incluido estos metadatos al registrar sus *datasets*.

### Procesamiento de datos

En esta etapa se procesan los recursos de datos descargados en la etapa anterior. Ver figura 4.3 donde se muestra la secuencia de forma gráfica. Para cada archivo cuyo nombre comienza con *table*, se intenta leer con codificación utf-8. Si esta lectura falla, se detecta la codificación más probable utilizando la biblioteca chardet [22]. Si esta lectura nuevamente falla, se lanza un error y se descarta la tabla.

Los archivos que se leen correctamente son truncados a un número fijo de filas, las cuales son seleccionadas aleatoriamente. Además, se eliminan las columnas con encabezado *"unnamed"* ya que en general se trata de columnas vacías que no aportan contenido significativo.

En esta etapa, se obtienen medidas relacionadas a la cantidad de archivos leídos exitosamente, con error de codificación, con una extensión errónea y con tamaño diferente al que se especifica en la respuesta de la API de CKAN. Además se registran cuantas columnas, filas y celdas fueron procesadas.

### Procesamiento de metadatos

En esta etapa se procesan los recursos de metadatos descargados en la primer etapa. En la figura 4.4 se muestra la secuencia de forma gráfica. Para cada archivo cuyo nombre comienza con *potential_metadata*, se detecta el tipo de archivo. En caso de que el formato detectado no sea JSON, se descarta el recurso.
Durante este procesamiento, se obtienen medidas relacionadas a la cantidad de archivos cuyo tipo de archivo no coincide con la extensión, y se corrige al almacenarlo. Además, se obtienen medidas de la consistencia del tamaño de archivo que se muestra en la API de CKAN y los que poseen un encoding no reconocido.

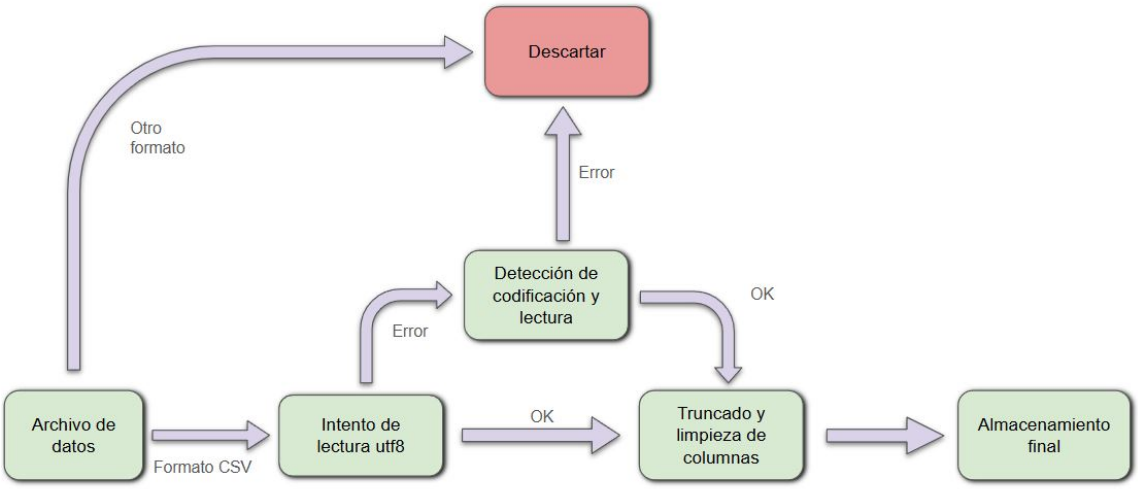


Figura 4.3: Secuencia de preprocesamiento de datos

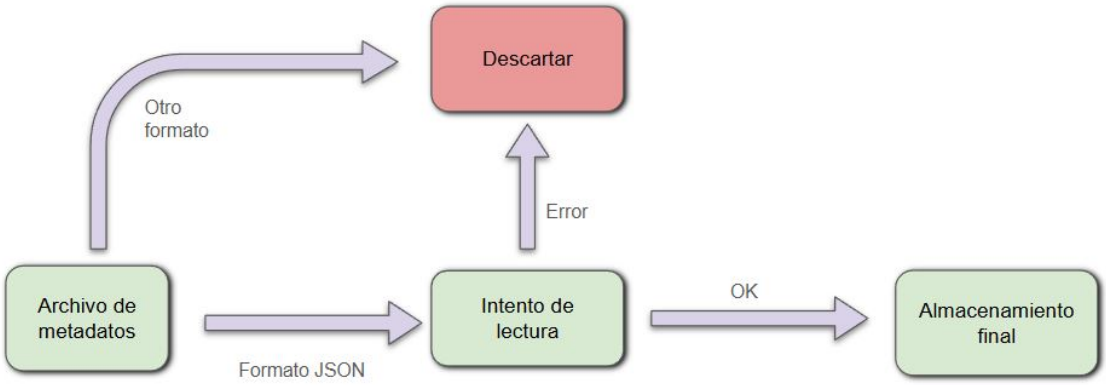


Figura 4.4: Secuencia de preprocesamiento de metadatos

### Selección de datasets y metadata

Una vez procesados los recursos de datos y metadatos, se realiza una selección que asocia una única tabla y su metadata correspondiente a cada *package*. En la figura 4.5 se muestra un diagrama

con el proceso. Este paso es necesario, ya que el Catálogo de Datos Abiertos no proporciona una relación explícita entre las tablas y sus metadatos.

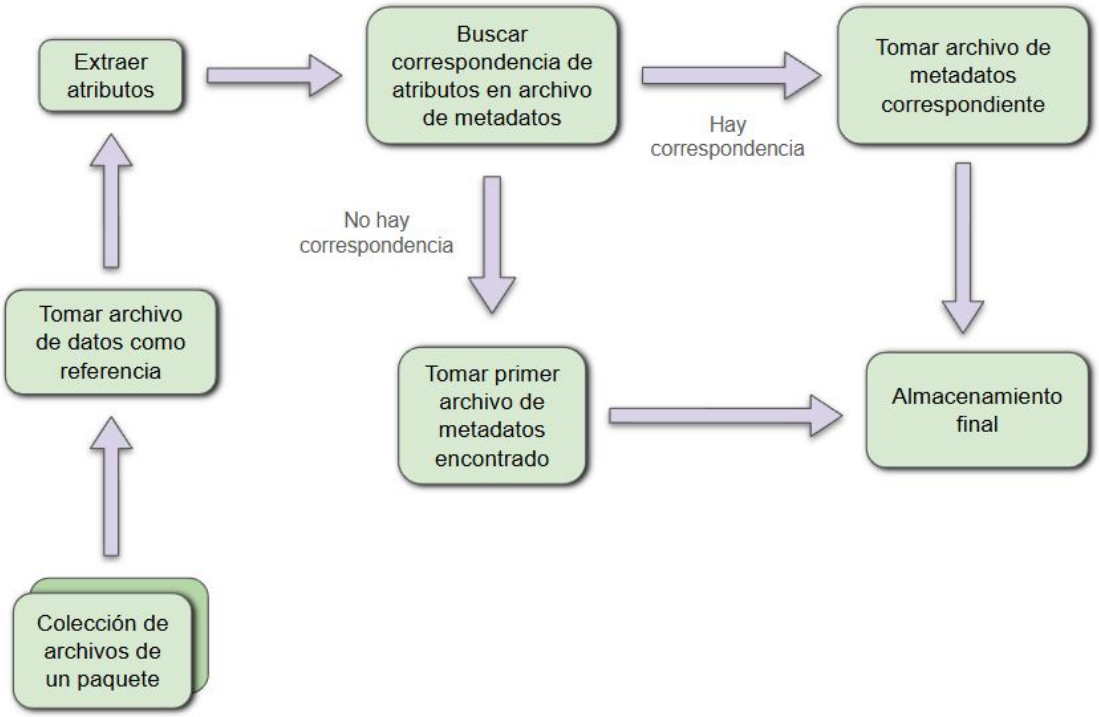


Figura 4.5: Secuencia de selección de recursos

El proceso comienza seleccionando un archivo de tabla o metadatos como punto de referencia, extrayendo sus atributos. Luego, se recorre el resto de los archivos del *package* en busca de aquel cuyo conjunto de atributos coincida con el del archivo inicial. Una vez encontrada la correspondencia, se almacena la tabla seleccionada con su metadata en el directorio correspondiente al *package*, asegurando así la consistencia de los datos para análisis posteriores.

De esta forma, se obtiene para cada *package*, un archivo de tabla en formato CSV, un archivo de metadata en formato JSON (en el mejor de los casos) y el archivo de información adicional, que se va actualizando a lo largo del procesamiento, eliminando o agregando recursos según corresponde.

## 4.3. Limpieza de datos

Con las medidas recopiladas en las fases de descarga, preprocesamiento y selección de datos, se evaluó la calidad de los conjuntos de datos del catálogo. Si bien se abarcó la totalidad de los

conjuntos presentes en el catálogo, no se descargó la totalidad del mismo debido a las restricciones impuestas, como el tamaño de los archivos, el número máximo de recursos por conjunto de datos y filtros basados en la extensión o el *encoding* de los archivos.

En la etapa inicial de descarga, se identificaron 2546 *packages* únicos dentro del catálogo, de los cuales se descargaron 2857 archivos de datos en formato CSV. Sin embargo, 23 archivos no pudieron ser recuperados debido a errores en la API de CKAN o enlaces incorrectos. Entre los archivos descargados, se encontraron diversas inconsistencias:

- 40 recursos fueron catalogados como CSV, pero contenían formatos incompatibles.
- 15 archivos tenían caracteres que impidieron la detección de su encoding.
- 8 archivos tenían un tamaño superior a 42MB, incumpliendo lo especificado por la metadata del catálogo.

En cuanto a los archivos de metadatos en formato JSON, se descargaron un total de 2465, con 22 fallas en la descarga. Entre los obtenidos:

- 121 archivos poseían una extensión incorrecta (no eran JSON).
- 11 archivos tenían problemas de encoding.

Estos resultados muestran no solo que la cantidad de archivos de metadatos descargados fue menor que la de archivos de datos, sino que los problemas de calidad fueron más frecuentes en los archivos de metadatos.

Tras las fases de preprocesamiento y selección de los datos, se conservaron un total de 2350 *packages*. Se descartaron aquellos que no contaban con al menos un recurso CSV válido. En cuanto a los que se conservaron:

- 202 no contaban con un archivo de metadatos correspondiente. Además, los archivos de metadata que habían no eran muy descriptivos pues de un total de 9586 columnas, se encontraron 5873 columnas con descripciones que contenían menos de 3 palabras (61.2 %).
- 70 archivos de datos no contenían una descripción en texto. Y otros 21 contenían una descripción con menos de 3 palabras de largo.

Este estudio evidencia oportunidades de mejora en la calidad y el nivel de detalle de la información disponible en el catálogo. En cuanto a los metadatos, la mayoría de las descripciones del conjunto de datos y de sus columnas se limitan a nombres y no brindan información suficiente para comprender de qué trata sin necesidad de descargar los archivos y revisarlos manualmente. Esta limitación dio paso a evaluar alternativas para enriquecer estos metadatos con descripciones más detalladas aprovechando el contexto del conjunto de datos como información del *package* obtenida desde la API de CKAN y filas del recurso CSV.

# Capítulo 5

# Diseño e implementación de la herramienta propuesta

En este capítulo, se presenta el diseño y los detalles de la implementación de la herramienta propuesta, que incluyen mejoras realizadas a la solución de base y los nuevos componentes que se incorporaron.

En la sección 5.1, se presenta el diseño de la herramienta final. Esta herramienta consiste en un *pipeline* basado en las herramientas identificadas durante la investigación, que integra una serie de procesos automatizados para inferir relaciones sintácticas y semánticas sobre los datos y generar o enriquecer los archivos de *metadata*.

En la sección 5.2, se realiza un análisis sobre la solución de base y se presentan las consideraciones que se tuvieron en cuenta para su adaptación y aplicación al catálogo de datos. Por otro lado, en la sección 5.3, se especifican mejoras y correcciones realizadas sobre las herramientas de la solución de base.

En la sección 5.4, se detalla la implementación de un proceso de generación de metadatos con descripciones enriquecidas a nivel de tabla y columna de los datos del catálogo. Esta implementación surge como respuesta a los problemas de calidad de datos detectados en el capítulo anterior.

En la sección 5.5, se describen las salidas del sistema que integran los resultados obtenidos de las herramientas aplicadas: una funcionalidad que permite realizar consultas en lenguaje natural sobre el catálogo y la generación de un grafo RDF que permite la navegación de datos.

Por último, en la sección 5.6 se detallan las tareas requeridas para el mantenimiento del sistema, abarcando los casos de ingesta, actualización y borrado de conjuntos de datos.

## 5.1. Pipeline

La solución final, como se muestra en la figura 5.1, integra tres de las herramientas de la solución de base que, a grandes rasgos, detectan relaciones sintácticas y semánticas entre los conjuntos de datos. Además, se incorporó un nuevo modulo que se encarga de generar descripciones enriquecidas, respondiendo al problema de calidad de metadatos detectado en el capítulo anterior. Finalmente, como salidas del sistema, se proveen: un mecanismo de consulta en lenguaje natural, y un grafo RDF con las relaciones inferidas entre los datos.

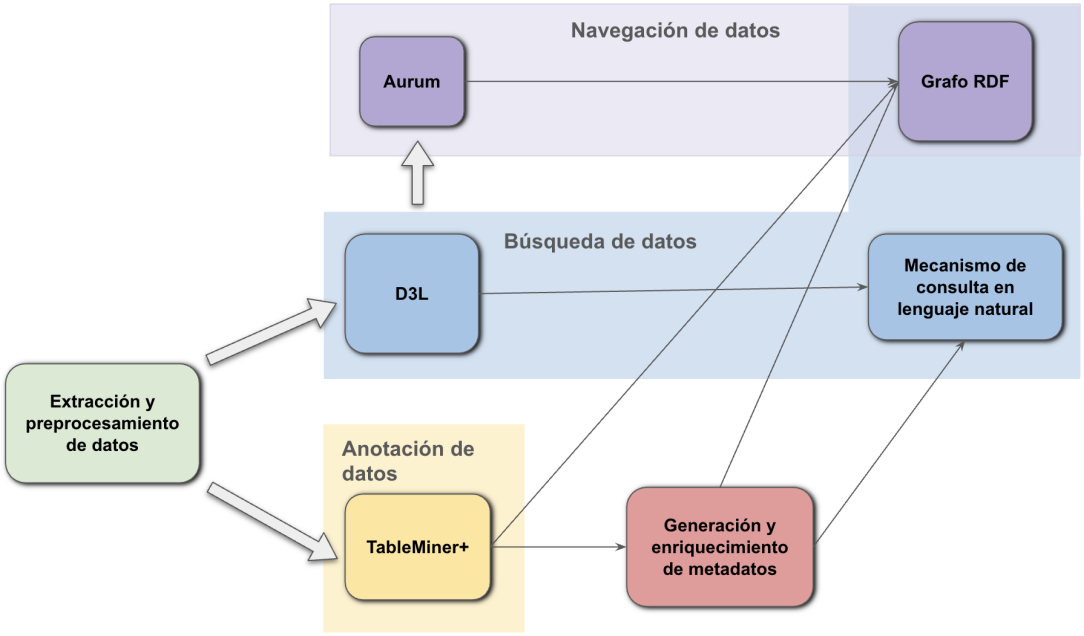


Figura 5.1: *Pipeline* de la herramienta.

El proceso comienza con la extracción, preprocesamiento y selección de datos del Catálogo de Datos Abiertos de Uruguay, como se detalla en 4.2. Para cada paquete descargado, idealmente se almacenan tres archivos: un archivo de datos en formato CSV (requerido), uno de metadatos en formato JSON y uno con información contextual del paquete, también en formato JSON.

Una vez seleccionados los conjuntos de datos, se ejecutan las herramientas de análisis en *batch*: en primer lugar, se generan los índices LSH como parte de D3L. Estos índices permiten posteriormente ejecutar Aurum, que construye un grafo identificando relaciones sintácticas entre columnas de distintos *datasets* mediante similitudes en los nombres y valores de los atributos.

En paralelo, se ejecuta TableMiner+, que realiza dos tareas principales. Primero, detecta el tipo sintáctico de cada columna y luego, aquellas columnas clasificadas como *named_entity* se anotan con conceptos de Wikidata, lo que permite establecer relaciones semánticas entre conjuntos de datos.

Finalmente, a partir del contexto del *package* y la salida de TableMiner+, se genera un archivo de metadatos enriquecido. Este archivo contiene una descripción detallada del *dataset* y de todas sus columnas, incluyendo su tipo sintáctico y, cuando está disponible, su tipo semántico.

Como salida del sistema, se ofrecen dos modalidades interactivas de exploración de los datos, según el perfil del usuario. Por un lado un grafo RDF con las descripciones generadas y las relaciones sintácticas y semánticas inferidas, y por otro, un mecanismo de consulta en lenguaje natural.

A continuación, se utilizan las salidas del sistema para recrear el caso de estudio presentado en la sección 1.1 como motivación del trabajo, donde se considera el caso de un analista de datos que desea investigar las desigualdades de género en los altos cargos de las instituciones estatales.

El analista comienza confirmando si existen datos relacionados a esta temática dentro del catálogo. Para esto hace uso del mecanismo de consulta en lenguaje natural, introduciendo la consulta *"Datos estadísticos de mujeres en altos cargos"*, obteniendo la respuesta que se muestra en la figura 5.2.

```
Para obtener información sobre mujeres en altos cargos, te recomiendo consultar los
siguientes conjuntos de datos:

- Distribución porcentual de las personas que ocupan altos cargos en entes autónomos
del Estado según sexo por ente. Total país: Este conjunto de datos proporciona
información detallada sobre la distribución de mujeres y varones en altos cargos en
diferentes entes autónomos del Estado. Puedes encontrar más información en:
https://catalogodatos.gub.uy/dataset/bf052edb-b02f-4699-aff5-c70c3d46ddff/resource/
5a933ce9-ba9c-4494-ac73-d2c57bd3302b/download/12084_distribucion_porcentual_de_las_
personas_que_ocupan_altos_cargos_en_entes_autonomos_del_est.csv
```

Figura 5.2: Respuesta del mecanismo de consulta en lenguaje natural

El analista toma este resultado como punto de partida para iniciar la navegación a través de *datasets* relacionados. Para esto, utilizando GraphDB[1] busca el *dataset* obtenido en la consulta anterior. En la figura 5.3 se muestra dentro del recuadro el *dataset* resultado. Luego expandiendo en la columna "Género", se obtienen otros *datasets* referentes a distribución de cargos importantes y relacionados al nivel educativo según sexo. Estas relaciones fueron inferidas automáticamente por Aurum, y son útiles para descubrir temas donde enfocar la investigación. Si se expande el *dataset* que habla de nivel educativo se pueden encontrar nuevos enfoques a analizar.

Además se podría iniciar la navegación a partir de una consulta SPARQL que integre otras características por ejemplo una organización que haya publicado este tipo de información, fechas, u otras palabras claves presentes en la descripción de los *datasets*.

[1]Base de datos de grafos semánticos que permite almacenar, organizar y gestionar datos enriquecidos.

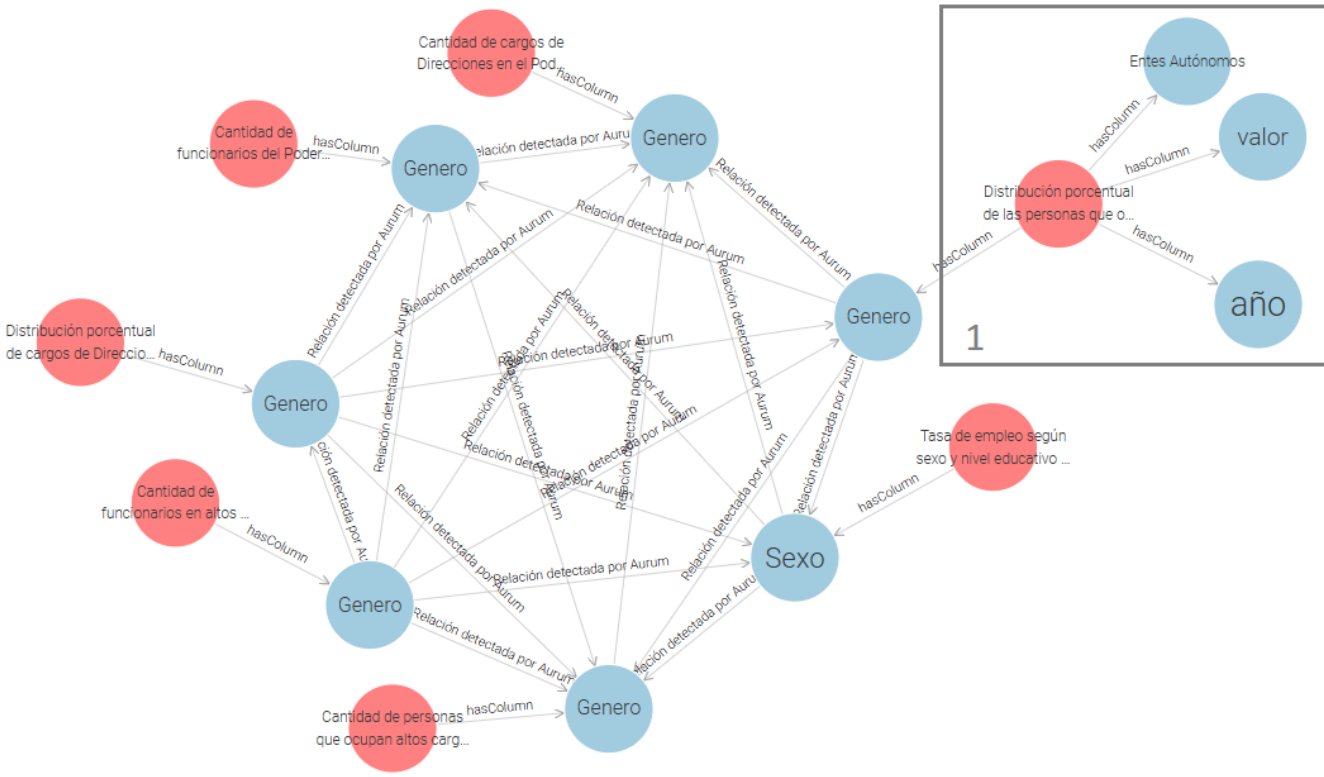


Figura 5.3: Grafo extraído de la consulta en GraphDB

## 5.2. Análisis de la solución de base

Como se detalla en 3.5, la solución de base consiste en un sistema basado en implementaciones simplificadas de D3L, Aurum, Tableminer+ y Starmie. Analizando los requerimientos específicos del caso de estudio (el Catálogo de Datos Abiertos del Gobierno), cada una de estas herramientas ofrecía ventajas particulares:

- **D3L:** Capacidad para realizar distintos tipos de búsqueda de datos: dado un *dataset*, encontrar similares; dada una columna, encontrar tablas relacionadas; dada una palabra clave, localizar *dataset* relevantes. Además, fue diseñado específicamente para *data lakes*, utilizando LSH, lo que le permite soportar grandes volúmenes de datos, característica fundamental en el contexto del catálogo.
- **Aurum:** Uno de los requerimientos principales desde el inicio del proyecto fue la posibilidad de explorar visualmente las relaciones entre los datos. Aurum proporciona este tipo de navegación basada en grafos identificando similaridad en nombres y valores de atributos. Además, se reutilizan los índices LSH de nombres y valores de atributos de D3L, como las *signatures* de

Aurum, y luego la construcción de relaciones entre columnas del grafo se basa en la similaridad según estos índices.

- **TableMiner+:** Mientras que D3L y Aurum se centran en identificar similitudes sintácticas, TableMiner+ introduce un nivel de análisis semántico. Esto es fundamental en el caso del catálogo ya que los datos provienen de múltiples fuentes y, con frecuencia, un mismo concepto puede aparecer representado de diferentes maneras.

Finalmente, la elección de estas herramientas no solo se basó en sus capacidades técnicas y alineación con los objetivos del proyecto, sino también en su disponibilidad como código abierto, lo que permitió partir de implementaciones ya existentes en lugar de desarrollar soluciones desde cero. Esto optimizó el tiempo de trabajo, permitiendo enfocarse en la adaptación e integración de las herramientas en el contexto específico del catálogo, así como en la identificación y mejora de sus limitaciones.

### 5.2.1. Otras consideraciones

Algunas herramientas implementadas en la solución base eran versiones simplificadas que omitían funcionalidades de las implementaciones originales. Estas simplificaciones se mantuvieron, y las funcionalidades no fueron incorporadas debido a que excedían el tiempo disponible para el proyecto.

En esta sección se detallan las razones detrás de la decisión de descartar herramientas completas, omitir funcionalidades específicas o no implementar ciertos componentes de las herramientas seleccionadas.

#### Aurum

La versión simplificada de Aurum, implementada en el la solución de base, se limita a la construcción del EKG, omitiendo las funcionalidades de mantenimiento y consulta. El mantenimiento del grafo no fue implementado por restricciones de tiempo, pero se diseñó una forma alternativa de mantenimiento que se detalla en la sección 5.6. Por otro lado, la funcionalidad de consulta se implementa de una forma alternativa detallada en la sección 5.5.1.

#### TableMiner+

Como se detalla en la sección 3.3, la implementación original de TableMiner+ aborda tres tareas fundamentales de anotación: enlazar menciones de entidades en celdas con entidades de referencia, anotar columnas con conceptos semánticos e identificar las relaciones semánticas entre columnas de una misma tabla.

En particular, la tarea de inferir relaciones semánticas entre columnas de una misma tabla, no estaba implementada en la versión de TableMiner+ de la solución de base y no fue agregada por motivos de alcance y tiempo del proyecto.
Por otro lado, en el caso del catálogo, la funcionalidad más relevante resultó ser la de anotar las columnas con un tipo semántico, ya que esto permite identificar de qué trata cada columna e identificar posibles relaciones entre columnas con el mismo tipo semántico, que formen parte de distintas tablas. En este contexto, la anotación de las celdas con entidades, si bien se utiliza para inferir el concepto asociado a la columna, no resultó útil como anotación de celda en sí. Muchas

de las instancias de las celdas no estaban representadas en la base de conocimiento debido a su especificidad y a que pertenecen al dominio uruguayo, lo que limitó su aplicabilidad. Por esta razón, no se incluyó la anotación de celdas con entidades en los metadatos generados.

#### Starmie

En la implementación original de la solución de base, Starmie se utilizaba como sistema de inferencia de esquemas en la última etapa del proceso. Aunque esta etapa sería útil para abordar los problemas de descubrimiento y exploración de datos, por restricciones de tiempo la implementación de esta herramienta no fue analizada en profundidad como las anteriores. El tiempo que se habría invertido en el análisis de esta herramienta se utilizó en su lugar para avanzar en otros objetivos que surgieron a lo largo del proyecto.

## 5.3. Mejoras de la solución de base

En primer lugar, se realizó una revisión completa del código, en la que se analizaron las clases utilizadas, los métodos principales de cada una y el flujo general de la herramienta. Simultáneamente, se desarrolló un diagrama que representaba estos elementos clave, detallando la funcionalidad, las interacciones y relaciones entre clases y métodos. Se adoptó esta aproximación porque para poder realizar aportes significativos, ya fueran mejoras o nuevas implementaciones, era fundamental contar con un conocimiento profundo del código.

### 5.3.1. TableMiner+

En TableMiner+, se identificaron diversas mejoras para optimizar su efectividad, aunque no todas pudieron ser implementadas debido a limitaciones de tiempo y recursos. Además, durante la ejecución y el análisis visual de los resultados, se detectaron y corrigieron múltiples errores, mejorando la precisión y coherencia en la anotación semántica de las tablas. En las siguientes secciones se detallan todas las mejoras propuestas y los errores corregidos.

#### Manejo del error 429 (Too Many Requests) en consultas a Wikidata

Durante la depuración del código se detectó un problema recurrente: muchas consultas al *endpoint* remoto de Wikidata devolvían el error 429 (**Too Many Requests**). Este error ocurría cuando se excedía el límite de solicitudes permitido por el servicio. En el flujo original, las consultas que generaban este error se trataban como si no existieran entidades asociadas a la entrada, lo que afectaba significativamente los resultados al introducir falsos negativos en las respuestas.

Además, estos errores generaban inconsistencias en los resultados de distintas ejecuciones de TableMiner+, ya que la consulta específica que fallaba podía variar en cada iteración, afectando la capacidad de reproducir experimentos y realizar análisis consistentes.

Para abordar este problema, se implementó un mecanismo de reintentos. Este mecanismo permite que, en caso de recibir un error 429, la consulta se reintente hasta tres veces. Si después del tercer intento, el error persiste, entonces se considera que no hay resultados disponibles para esa

consulta. Esta solución demostró ser efectiva, ya que en la mayoría de los casos, el problema se resolvía en el segundo intento.

#### Caché para almacenar resultados de consultas a la base de conocimiento

En las primeras ejecuciones de TableMiner+, incluso utilizando un solo *dataset*, el tiempo de procesamiento era excesivo y el proceso no llegaba a completarse. Este retraso se debía principalmente a la gran cantidad de consultas realizadas al *endpoint* de la base de conocimiento. Además, en las columnas *named-entity*, donde los valores de las celdas tienden a repetirse, se generaban múltiples consultas idénticas. Por ejemplo, en la tabla 5.1, la columna "LEMA" contiene solo cuatro valores únicos y veinte filas. En este caso, la mayoría de las consultas a la base de conocimiento eran innecesarias y redundantes.

| DEPARTAMENTO | SERIES | LEMA | CANTIDAD_VOTOS |
|---|---|---|---|
| CO | NHB | Frente Amplio | 7 |
| PA | KCB | Partido Colorado | 3 |
| SJ | OGC | Partido Nacional | 2 |
| CA | CMG | Partido Nacional | 4 |
| SO | MCA | Partido Nacional | 1 |
| SO | NFA | Partido Colorado | 14 |
| MO | SJA | Frente Amplio | 9 |
| MA | DAA | Partido Nacional | 2 |
| MO | BMB | Frente Amplio | 2 |
| LA | SAA | Frente Amplio | 14 |
| PA | KEA | Partido Nacional | 3 |
| MO | BCD | Partido Independiente | 20 |
| MO | BSB | Frente Amplio | 9 |
| PA | KAA | Partido Nacional | 2 |
| MO | BBD | Frente Amplio | 17 |
| SI | OAA | Partido Nacional | 4 |
| RV | HBB | Frente Amplio | 1 |
| FD | QAA | Partido Colorado | 6 |
| RV | HAB | Partido Nacional | 4 |
| MO | BBA | Frente Amplio | 1 |

Tabla 5.1: Tabla extraída del Catálogo de datos Abiertos de Uruguay [2]

Para abordar este problema, se implementó una caché que almacena un diccionario que mapea cada consulta a sus resultados correspondientes. En la figura 5.4 se muestra su funcionamiento en conjunto con el mecanismo de reintentos de la sección anterior.

De esta manera, antes de realizar una consulta a la base de conocimiento, se verifica si el resultado ya está presente en la caché. Si es así, se devuelve directamente el resultado en O(1); de lo contrario, se realiza la consulta a la base de conocimiento. La mejora en el rendimiento es considerable: al ejecutar el método para buscar una entidad en la base de conocimiento a partir del contenido de una celda, sin utilizar la caché, cada consulta tarda aproximadamente 1.5 segundos. Sin embargo, una vez que los resultados se almacenan en la caché, las consultas posteriores se completan instantáneamente.

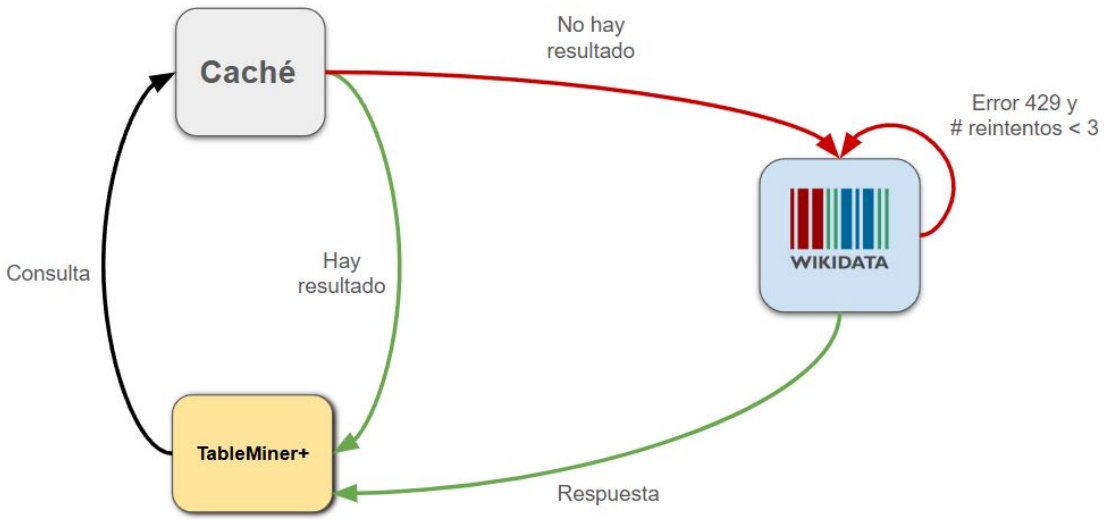


Figura 5.4: Caché y mecanismo de reintentos implementado

Analizando las consultas realizadas durante la etapa de desambiguación preliminar de celdas (detallada en 3.3.2) para obtener entidades asociadas al contenido de las celdas, volviendo al ejemplo de la columna "LEMA" de la tabla 5.1, se realizan 20 consultas (una por cada celda). Sin el uso de caché, se realizarían 20 consultas independientes a la base de conocimiento, lo que requeriría un tiempo total de aproximadamente 30 segundos. Con la caché implementada, solo se realizarían 4 consultas iniciales, y el resto de los resultados se recuperarían de la caché, lo que reduciría el tiempo total a aproximadamente 6 segundos. Esto demuestra una reducción significativa en el tiempo de ejecución, gracias a la eliminación de consultas redundantes y la optimización en el acceso a los datos.

### Mejoras en tiempos de respuesta

Siguiendo con el objetivo de reducir el tiempo de las consultas directas a la base de conocimiento, se consideró la opción de utilizar un motor de base de datos de grafos local. En particular, se utilizó

Blazegraph [5], que permitió realizar consultas mucho más rápidas al eliminar la dependencia de interacciones constantes con servicios remotos.

En esta fase, se utilizaba DBpedia[2] como base de conocimiento. Se descargaron los archivos principales del core de DBpedia en inglés con los predicados que se utilizaban en las consultas de TableMiner+ a la base de conocimientos. En total eran 135 millones de triplas y ocupaban aproximadamente 25 GB de espacio en disco. Las diferentes triplas fueron descargadas en archivos por separado según el predicado, por ejemplo, se tenía un archivo con todas las triplas del predicado *label*, otro archivo para las del predicado *abstract* y de forma similar para el resto.

Uno de los principales desafíos fue construir índices eficientes para realizar búsquedas de texto, necesarias para encontrar entidades cuyas etiquetas coincidieran con el contenido de una celda. Dada la magnitud del grafo, realizar estas búsquedas de manera lineal era inviable, lo que llevó a explorar diferentes enfoques para la indexación. Inicialmente, se intentó utilizar Elasticsearch[3] para construir los índices de búsqueda textual, debido a su reconocida capacidad para manejar comparaciones textuales complejas. Sin embargo, su integración con Blazegraph resultó ser costosa en términos de tiempo, recursos computacionales y complejidad de desarrollo, lo que hizo que este enfoque no fuera viable.

Finalmente, se optó por utilizar los índices nativos de Blazegraph para simplificar el proceso. Aunque esta solución también fue intensiva en tiempo y recursos, demorando aproximadamente un día en construir toda la estructura de datos, representó un compromiso razonable entre simplicidad y funcionalidad. Sin embargo, fue necesario realizar un preprocesamiento sobre el subconjunto de triplas que se descargó de DBpedia dado que en Blazegraph hay restricciones de tamaño de archivo para cargar nuevas triplas en la estructura de datos subyacente. Esto significó que cada archivo de triplas se tuvo que dividir en fragmentos más pequeños, cada uno conteniendo hasta 250.000 triplas. Esto permitió no solo cargar de a tandas más pequeñas las triplas al *journal* de Blazegraph, sino que también cumplió el rol de un mecanismo de recuperación, dado que en caso de fallos era más sencillo comenzar desde un determinado fragmento que volver a cargar el archivo desde cero.

Una vez finalizado este proceso, las consultas más complejas, como las búsquedas de texto basadas en el contenido de las celdas, se ejecutaban en promedio en 150 ms, lo que representaba una mejora de 10 veces en comparación con las consultas realizadas a un *endpoint* remoto. Por otro lado, las consultas más directas, como aquellas donde ya se dispone de la URI y solo se buscan conceptos asociados, se ejecutaban en 50 ms, siendo aproximadamente 30 veces más rápidas que las realizadas al *endpoint*.

Sin embargo, tras revisar herramientas actuales de anotación de datos en el contexto de la competencia SemTab [26], y observar que la mayoría de las herramientas emplean Wikidata, se analizó la posibilidad de hacer el cambio de DBpedia a Wikidata. Este cambio resultó sencillo, ya que ambas opciones estaban implementadas en el notebook; solo fue necesario ajustar un parámetro para seleccionar la fuente de datos y añadir la funcionalidad de caché para las consultas a Wikidata.

[2]Proyecto de base de conocimiento que extrae información estructurada desde Wikipedia y la pone a disposición en la web en formato RDF, siguiendo los principios de la Web Semántica

[3]Elasticsearch es un motor de búsqueda y analítica distribuido, un almacén de datos escalable y una base de datos de vectores.

Al realizar pruebas con Wikidata (ver Anexo A.3.1), se observó una mejora significativa no solo en la precisión, sino también en el tiempo de respuesta, lo que motivó la decisión de adoptar esta base de conocimiento. Sin embargo, al intentar construir el subgrafo local en Blazegraph para Wikidata, surgió un obstáculo: Wikidata solo permite descargar un dump completo, que pesa aproximadamente 1TB. Este volumen de datos excedió las capacidades de almacenamiento disponibles, lo que dificultó su implementación como subgrafo local.

Aunque la estrategia inicial basada en Blazegraph con un subgrafo local de DBpedia demostró ser efectiva en términos de tiempo de consulta, las limitaciones de almacenamiento y la imposibilidad de manejar un dump completo de Wikidata hicieron inviable la implementación de un grafo local para esta nueva fuente de datos. No obstante, esta estrategia queda identificada como una posible mejora futura, ya que contar con un grafo local optimizado podría combinar la precisión de Wikidata con tiempos de respuesta significativamente menores, siempre que se disponga de los recursos de almacenamiento y procesamiento adecuados para su implementación.

### Detección de errores de anotación

Una vez optimizada la etapa de procesamiento, se procedió a realizar un análisis visual de los resultados obtenidos. Para ello, se utilizó la tabla personalizada de la figura 5.2 diseñada con columnas y celdas que contenían entidades conocidas que TableMiner+ podría anotar (por ejemplo, "auto", "película", "género").

| **Nombre** | **Actor** | **Género** | **Año** |
|---|---|---|---|
| The Shawshank Redemption | Tim Robbins | Drama | 1994 |
| The Godfather | Marlon Brando | Crime | 1972 |
| The Dark Knight | Christian Bale | Action | 2008 |
| The Godfather: Part II | Al Pacino | Crime | 1974 |
| The Lord of the Rings: The Return of the King | Elijah Wood | Adventure | 2003 |

Tabla 5.2: Tabla personalizada para probar resultados

En la ejecución de TableMiner+ con esta tabla, sabiendo los resultados esperados, se identificaron varios errores que se detallan a continuación.

**Error en actualización de puntajes de conceptos:** Al ejecutar TableMiner+ con esta tabla, la columna "nombre", que contenía nombres de películas, fue anotada con el tipo semántico **"musical work / composition"** (Q105543609). Como se detalla en la sección 3.3, la anotación semántica de una columna se genera a partir de los tipos semánticos de las entidades identificadas para cada celda. Sin embargo, durante el proceso de depuración del código, se observó que, al realizar la búsqueda por el contenido de cada celda, la mayoría devolvían entidades cuyo tipo semántico era **"film"** (Q11424). El tipo **"musical work / composition"** solo se asignaba a la celda **"The Shawshank Redemption"**, donde las entidades identificadas incluían **"The Shawshank Redemption"** (Q172241 de tipo **"film"** y Q110239360 de tipo **"musical work/composition"**).

Este comportamiento generó la duda sobre cómo y por qué el concepto **"film"**, siendo que aparecía en la mayoría de celdas, era desplazado por **"musical work / composition"**. Tras analizar el código, se identificó el problema en la fase de actualización (detallada en 3.3.3). Durante esta etapa, al recalcular los puntajes de los conceptos, únicamente se consideraban los conceptos ganadores. En iteraciones posteriores, todos los conceptos eran reevaluados, sin embargo, los conceptos ganadores previamente identificados en la fase de aprendizaje (detallada en 3.3.2) no mantenían su ventaja, ya que los puntajes no se actualizaban correctamente. Esto causaba que los conceptos inicialmente ganadores perdieran relevancia frente a otros conceptos.

Si bien es correcto actualizar los puntajes para reflejar los cambios introducidos durante la etapa de desambiguación, considerando todas las celdas procesadas y la representación del dominio, el error radicaba en que solo algunos conceptos eran actualizados, dejando en desventaja a los conceptos inicialmente ganadores. Por ejemplo, en el cálculo de *conceptInstanceScore* (detallado en 3.3.5), el denominador es la cantidad total de celdas anotadas; sin embargo, en la implementación original, los conceptos cuyo puntaje se actualizaba correctamente se normalizaban respecto a la cantidad total de celdas, mientras que aquellos que no fueron ganadores mantenían su puntaje dividido únicamente por la cantidad de celdas procesadas en el momento en que fueron identificados. Esta inconsistencia generaba un sesgo en la selección de conceptos finales.

Una vez solucionado este problema, se aseguró que los puntajes de todos los conceptos se actualizaran de manera uniforme, lo que permitió obtener resultados precisos para cada celda y garantizar que el tipo semántico asignado reflejara fielmente la naturaleza de los datos analizados.

**Búsqueda de conceptos asociados a todas las entidades encontradas para una celda:** Por otro lado, en el análisis de los resultados se observó que la columna "género" fue anotada incorrectamente con el tipo **"social issue"** (Q1920219). Al depurar el código, se identificó el problema en la etapa de clasificación preliminar de columnas durante el proceso de desambiguación inicial. Como se detalla en 3.3.2, en esta etapa se recorre una muestra reordenada de las celdas de la columna para seleccionar entidades ganadoras asociadas a cada celda y, a partir de estas, determinar conceptos candidatos (tipos semánticos) para anotar la columna.

En el caso específico de la tabla 5.2, la muestra seleccionada para la columna "género" incluyó las celdas "Crime" (fila 1) y "Crime" (fila 3), tras lo cual el proceso de clasificación preliminar convergió. Para la entrada "Crime", Wikidata devolvió las siguientes entidades: **"Crime"** (Q3697152) con el tipo semántico **"musical group"**, **"crime"** (Q83267) con el tipo semántico **"social issue"**, y **"crime film"** (Q201658) con el tipo semántico **"film genre"**. Sin embargo, al seleccionar la entidad ganadora, el algoritmo utiliza un puntaje basado en la coincidencia sintáctica entre el contenido de la celda y la etiqueta de la entidad. Esto favoreció a las entidades con etiquetas idénticas al contenido de la celda, descartando así la entidad **"crime film"**, cuyo tipo semántico era el correcto según la información completa de la tabla.

Además, en la siguiente etapa, como se detalla en 3.3.2, únicamente se consideran entidades que tengan algún concepto asociado que coincida con alguno de los conceptos ganadores identificados en la etapa de clasificación preliminar. Como el concepto **"film genre"** había sido descartado en la etapa previa, no se toma en cuenta en el procesamiento de las celdas restantes, aunque apareciera en todas ellas.

Para resolver este problema, se implementó una modificación en el algoritmo para que, en lugar de limitarse a los conceptos de las entidades ganadoras, se incluyeran los tipos semánticos de todas las entidades devueltas por Wikidata en la consulta para una celda. Dado que Wikidata devuelve resultados bien ordenados y se estableció un máximo de tres entidades por consulta, esta mejora garantiza que no se descarten conceptos relevantes como "film genre" en este caso. Además, el impacto en el tiempo de procesamiento es mínimo debido al límite de resultados por consulta. Esta modificación responde a la observación de que la entidad con el nombre más similar al contenido de la celda no siempre es la más representativa en términos semánticos.

Tras las correcciones, la columna "nombre" fue correctamente anotada con el tipo **"film"** (Q11424), y la columna "género" con el tipo **"film genre"** (Q201658). Estos resultados coincidieron con las expectativas establecidas para la tabla diseñada específicamente, confirmando la efectividad de los ajustes realizados.

**Omisión de búsqueda de conceptos para entidades identificadas en la desambiguación:**

Durante la revisión realizada para detectar el problema detallado anteriormente, también se identificó un error en la etapa de Desambiguación Preliminar de Celdas (detallada en 3.3.2). En esta fase, una vez finalizada la Clasificación Preliminar de Columnas, el proceso se reinicia desde la primera celda de la columna.
El problema surge al determinar si se deben buscar conceptos asociados a la entidad ganadora. Para esto, se utiliza una condición que verifica si la celda ya fue marcada como "visitada" y si la columna ya tiene conceptos asociados. La idea es que si la celda ya fue visitada, no es necesario volver a buscar conceptos.

Sin embargo, cuando se visita una celda por primera vez en la fase de desambiguación preliminar, su índice de fila se marca inmediatamente como "visitado" tras encontrar entidades candidatas. Como en este punto ya hay conceptos asociados a la columna (por estar en la etapa de desambiguación), la celda cumple con ambas condiciones del "if", lo que provoca que el proceso omite la búsqueda de conceptos para la entidad ganadora de la celda.

Este error tiene un impacto significativo, ya que todas las entidades asociadas a celdas visitadas por primera vez en la fase de desambiguación quedan sin conceptos asignados.
Como consecuencia, al actualizar los puntajes de los conceptos en la fase de actualización, los conceptos de estas celdas no se toman en cuenta, sesgando el proceso.

En la práctica, esto equivale a considerar únicamente los conceptos obtenidos en la Clasificación Preliminar de Columnas, ignorando los conceptos que podrían haberse identificado en la Desambiguación Preliminar de Celdas, lo que limita la capacidad del sistema para mejorar la precisión de la anotación en iteraciones posteriores.

**Acumulación incorrecta de entidades candidatas durante el procesamiento de celdas**

Otro error identificado estaba relacionado con la selección de las entidades ganadoras para cada celda. Se descubrió que las entidades candidatas se acumulaban en una estructura de datos que no se reinicializaba correctamente al procesar cada celda. Esto provocaba que, al identificar la entidad

ganadora de una celda, también participaran como candidatas entidades provenientes de búsquedas realizadas para otras celdas procesadas anteriormente. Este comportamiento introducía ruido en el proceso de selección, afectando la precisión de los resultados.
La solución consistió en asegurar que la estructura que almacena las entidades candidatas se inicializara en vacío al comenzar el procesamiento de cada celda.

### 5.3.2. Manejo de valores faltantes en celdas

Al utilizar *datasets* provenientes del Catálogo de Datos Abiertos de Uruguay, surgieron nuevos desafíos.
En primer lugar, al analizar los datos descargados, se observó una gran cantidad de columnas con valores como "N/A" (no aplica), los cuales son considerados valores faltantes. Sin embargo, en la función que determina el tipo de cada columna para identificar columnas *named-entity*, estos valores no se identificaban adecuadamente. Aunque la función reconocía celdas vacías y acrónimos, los valores faltantes no se ajustaban a ninguno de estos casos, por lo que terminaban siendo clasificados erróneamente como *named-entity* y se les aplicaba TableMiner+ de la misma forma que a las otras columnas. Esto ocasionó que incluso columnas numéricas con una alta proporción de valores faltantes fueran etiquetadas como *named-entity*, lo cual evidentemente no era correcto.

Para solucionar este problema, se incorporó una lista denominada "*missing values*", que incluye valores como "N/A", "S/I", "S/R", entre otros. En estos casos, la columna se clasifica bajo el tipo "*other*". Luego, si una columna es clasificada como "*other*", se cuenta cuántos de sus valores corresponden a valores faltantes. Si la mayoría de estos valores son faltantes, la columna se reclasifica según el segundo tipo más frecuente encontrado en ella, para no perder la información relevante del tipo original de la columna.

#### Problemas del dominio de los datos del Catálogo

Se encontraron problemas específicos relacionados con las características del dominio. En este caso, el problema no radicaba en el funcionamiento intrínseco de la herramienta, sino en su aplicación a los datos del catálogo.

El correcto desempeño de TableMiner+ depende de la capacidad para identificar instancias en una base de conocimiento (en este caso Wikidata) y, a partir de los tipos semánticos asociados a las entidades encontradas para las celdas, anotar la columna con el tipo semántico predominante entre dichas entidades. Sin embargo, muchas tablas del catálogo contienen instancias altamente específicas y locales. Por ejemplo, una columna que enumera nombres de instituciones uruguayas como "Administración Nacional de Combustibles, Alcohol y Portland", enfrenta dificultades porque estas entidades suelen no estar representadas en Wikidata por su especificidad. En este caso, si bien el concepto "empresa" u "organización" existe en Wikidata, la ausencia de entidades específicas asociadas a las celdas, impide que la columna sea clasificada con un tipo semántico. Este problema no se presenta para entidades más generales, como los departamentos uruguayos, que sí cuentan con representación en la base de conocimiento.

Para abordar esta limitación, se implementó un mecanismo de *fallback* diseñado para mitigar los casos en los que la falta de anotación es resultado de la ausencia de entidades específicas en la base de conocimiento. Este mecanismo utiliza un LLM, que considera además el título de la columna

y el contexto del *dataset*, para generar una etiqueta que describe el tipo semántico general de los datos contenidos en la columna. En el anexo 3.3.2 se encuentra el *prompt* utilizado para generar el concepto.
Posteriormente, esta etiqueta es utilizada para buscar en Wikidata una clase que se corresponda con el concepto generado. Si se identifica un concepto, este se emplea para anotar la columna. Este último paso asegura que las anotaciones sean compatibles con un concepto existente en Wikidata. Vincular las columnas a Wikidata, asegura que cada entidad etiquetada cuenta con una definición precisa y estándar en la base de conocimiento, eliminando ambigüedades y facilitando la interoperabilidad entre sistemas que también utilizan esta base como referencia.

Una vez implementadas las mejoras y adaptaciones a la solución de base, se comenzó la implementación de un nuevo módulo, que consiste en la generación de descripciones enriquecidas a nivel de tabla y columna de los datos del catálogo. En la siguiente sección se detalla su implementación.

## 5.4. Generación de metadatos

Con el fin de abordar una de las oportunidades de mejoras en la calidad de los metadatos del catálogo, evidenciadas en la evaluación detallada en 4.3, se decidió integrar un nuevo módulo a la herramienta propuesta. Este módulo consiste en la generación de metadatos con descripciones del contenido general del *dataset* y de cada columna, a través de un LLM que recibe información contextual. En esta sección se detalla su diseño e implementación.

### 5.4.1. LLM utilizado

Se utilizó *LLaMA 3.2 Instruct*[4] como modelo de lenguaje para la generación de descripciones. Inicialmente, se comenzó con la versión 1B, pero esta presentó varias limitaciones de interpretación de instrucciones, entendimiento del contexto y errores ortográficos en las respuestas. En consecuencia, se decidió utilizar la versión 3B, que demostró un rendimiento superior. Esta mejora está directamente relacionada a la cantidad de parámetros que cada uno posee, observar que el 1B y el 3B del nombre de los modelos indican que estos tienen 1000 y 3000 millones de parámetros respectivamente, lo cual es un indicador directamente relacionado a la capacidad del modelo de comprender relaciones complejas entre las entradas y las salidas. No se utilizaron modelos aún más avanzados, con más parámetros debido a que se tenía que adaptar al limitado entorno de trabajo.

### 5.4.2. Diseño del procedimiento

El objetivo de esta etapa es generar una descripción del conjunto de datos, y un archivo de metadatos similar al provisto en algunos casos por el catálogo (Ver anexo B.2). En este archivo se incluye para cada columna: la descripción generada, el tipo sintáctico (que se obtiene para la detección de la *subject column* detallado en 3.3.1) y el tipo semántico con el que se anotó la columna por TableMiner+ en caso de que haya habido coincidencias.

[4]Conjunto de LLMs generativos preentrenados y ajustados para seguir instrucciones, disponibles en tamaños de 1B y 3B.

Para poder generar los metadatos faltantes, inicialmente es necesario reconocer qué metadatos le faltan a cada conjunto de datos que está siendo procesado. Para ello se realiza una división de los conjuntos de datos entre:

- Conjuntos que tienen archivo de metadatos y descripción del conjunto de datos
- Conjuntos que solo tienen archivo de metadatos
- Conjuntos que solo tienen descripción
- Conjuntos que no tienen ninguna de las dos

Se utilizaron diferentes *prompts* siguiendo el enfoque de *in-context learning*, utilizando *few shots*[5] que fueron preparados utilizando conjuntos de datos encontrados en el catálogo de datos.
En la figura 5.4.2 se muestran las instrucciones que fueron comunes para todas las *prompts*, con el fin de adaptar el modelo al contexto de los datos utilizados.

```
[PROMPT]
Eres un asistente que ayuda en la desambiguación de tablas.
Toda la información pertenece al catálogo de datos abierto de Uruguay.

### Instrucciones
- Solo se debe generar como output descripciones detalladas y específicas.
- No uses frases genéricas como "No hay datos relevantes".
  Omitir en la respuesta todo lo que no sea una descripción.
- Solo responder con la descripción, ser lo más objetivo posible.
```

Figura 5.5: Instrucciones en común de los prompts

Luego, dependiendo de la información disponible, se utilizaron diferentes *prompts*:

1. **Generación de descripción:** Para los conjuntos de datos que no poseen descripción ni archivo de metadatos, se utiliza un *prompt* para generar su descripción enviándole al LLM: nombre del conjunto de datos que figura en el catálogo, nombre del archivo CSV que contiene los datos y algunas filas de la tabla. El *prompt* se encuentra en el anexo C.1.
2. **Generación de descripción usando archivo de metadatos existente:** Al igual que el *prompt* anterior, el objetivo de este es generar una descripción detallada del conjunto de datos, pero en este caso además se cuenta con un archivo de metadatos que contiene información sobre cada columna del CSV con los datos, incluyendo una descripción para cada columna y su tipo de dato. El *prompt* se encuentra en el anexo C.2.
3. **Generación de archivo de metadatos:** El objetivo de este *prompt* es generar una descripción para una columna en particular de la tabla que se está procesando. La información que se utiliza para este *prompt* incluye: nombre del conjunto de datos en el catálogo de datos, descripción general del conjunto de datos, nombre del archivo CSV y algunas filas de la tabla.

[5]Enfoque de aprendizaje donde un modelo aprende a hacer una tarea nueva a través de ejemplos.

En este caso, se harán tantas consultas al LLM como columnas tenga la tabla, de forma de que al final sea posible construir el archivo de metadatos en formato JSON. El *prompt* se encuentra en el anexo C.3.

La generación de metadatos se puede realizar secuencialmente iniciando con los conjuntos de datos que no poseen ninguna información. Una vez que se generan sus descripciones, estos pasan a ser de la categoría que posee descripción pero no archivo de metadatos. A continuación, se genera la descripción de cada una de las columnas de los archivos CSV, utilizando la descripción general del paquete de datos como contexto para el modelo. Esta información generada sobre cada columna del CSV se utiliza para generar los archivos de metadatos.
Por último, para generar las descripciones de los conjuntos de datos que tienen archivo de *metadata* pero no cuentan con descripción general del paquete de datos, se utiliza la información de *metadata* como contexto para el LLM.

## 5.5. Salidas del sistema

Hasta este punto se tienen las relaciones inferidas entre columnas de *datasets*, las anotaciones de columnas con conceptos de Wikidata y los archivos de metadatos enriquecidos para cada conjunto de datos.

A continuación se presentan las funcionalidades que se proveen como salidas del sistema para facilitar el descubrimiento y exploración de conjuntos de datos dentro del catálogo.

### 5.5.1. Construcción de grafo RDF

Si bien cada herramienta aporta información relevante, usarlas por separado hace que sea difícil visualizar cómo se conectan los datos. En el contexto del presente trabajo, un *dataset* puede estar relacionado con otro porque tienen columnas con nombres o valores similares (relaciones sintácticas), o porque comparten la misma entidad de Wikidata en sus anotaciones (relaciones semánticas), aunque los nombres de las columnas sean distintos. Tener un lugar donde estas conexiones sean visibles facilita descubrir información útil y entender mejor el contexto de cada conjunto de datos.

Además, contar con los metadatos enriquecidos de cada *dataset*, incluyendo su descripción generada, los tipos de datos de sus columnas y los enlaces a los archivos originales permite a los usuarios decidir qué *datasets* son realmente relevantes antes de descargarlos, en lugar de revisar manualmente cada archivo.

Para lograr esto, se construyó un grafo RDF que almacena todas las relaciones inferidas y metadatos generados. Tener toda esta información en un formato estructurado permite ejecutar consultas avanzadas, combinando diferentes criterios como fechas de publicación, relaciones con entidades de Wikidata o coincidencias estructurales entre columnas. Esto simplifica la exploración del catálogo, permitiendo descubrir datos relevantes más allá de lo que se lograría con una simple búsqueda por palabras clave.

**Estructura del grafo**

El grafo RDF está compuesto por diferentes tipos de nodos y relaciones, organizados de la siguiente manera:

- Nodos de *datasets*: Cada conjunto de datos en el catálogo es representado como un nodo en el grafo. Este nodo almacena metadatos como su título, una descripción detallada, la organización que lo publicó, la fecha de creación y un enlace al recurso original.
- Nodos de columnas: Cada columna dentro de un *dataset* se modela como un nodo propio, vinculado al *dataset* al que pertenece. Además de su nombre, cada columna contiene información sobre su tipo sintáctico, y su tipo semántico, obtenidos de la ejecución de TableMiner+.
- Relaciones sintácticas entre columnas: Se establecieron relaciones entre columnas de distintos *datasets* utilizando las aristas provistas por el grafo generado por Aurum.
- Relaciones semánticas con Wikidata: En el grafo, cada columna almacena su tipo semántico, lo que permite establecer conexiones automáticas entre columnas de distintos *datasets* que comparten la misma entidad. Esto facilita la identificación de datos que aunque su nombre difiera sintácticamente, representan la misma entidad.

Finalmente, se almacenó la información generada en un archivo RDF en formato Turtle, y utilizando la herramienta de visualización GraphDB fue posible navegar entre las relaciones inferidas, incluso partiendo de una consulta de interés en SPARQL.

Si bien este mecanismo proporciona una forma de explorar las relaciones inferidas, puede no ser la más adecuada para un usuario sin conocimientos técnicos. En la siguiente subsección se presenta una herramienta alternativa para este tipo de usuarios.

### 5.5.2. Consultas en lenguaje natural sobre los datos

Imaginando el caso de uso desde la perspectiva de un usuario que desea consultar datos estadísticos de analfabetismo, sería deseable que fuese tan fácil como introducir la consulta *Datos sobre Tasa de analfabetismo en Uruguay"* y se pueda obtener como respuesta recursos de interés, junto con información relevante de su contexto.

Para dar estas respuestas al usuario usando información que se tiene sobre el contexto de todos los conjuntos de datos que se procesaron, se utilizó un enfoque de RAG (detallado en 2.4.2). De esta forma, el LLM no solo cuenta con su conocimiento preentrenado sino que también se le puede brindar información adicional especializada.

El primer paso requerido es generar los *word embeddings* necesarios sobre la información que se posee sobre los conjuntos de datos, de forma de que posteriormente se pueda realizar una comparación en el espacio vectorial entre la consulta en lenguaje natural del usuario contra la *metadata* existente. En particular, para cada conjunto de datos se junta: título, descripción general y organización publicadora, para luego ser codificados en *word embeddings* y capturar su significado semántico.
El modelo utilizado para generar los *word embeddings*, se conoce por el nombre *all-mpnet-base-v2* y se destaca particularmente por su capacidad para capturar el significado semántico de oraciones,

siendo especialmente útil para tareas de recuperación de información [18].

A diferencia de otros métodos donde el modelo se entrena *end-to-end* para mapear directamente las consultas de los usuarios a sus respuestas, en este caso se adopta un enfoque híbrido que separa la codificación de la información preexistente (la *metadata* de los conjuntos de datos) de la codificación de la consulta en lenguaje natural. De este modo, solamente se necesita calcular los *embeddings* de la información de los datos una sola vez (o cuando surjan nuevas actualizaciones en la metadata), mientras que las consultas del usuario se codifican sobre la marcha.
Esto reduce notablemente el costo computacional en escenarios donde las consultas cambian con frecuencia, pues el cálculo de *embeddings* de la *metadata* —que puede ser muy extenso— se realiza de forma previa y permanece *cacheado*. Así, cuando un usuario ingresa una consulta, basta con codificarla con el mismo modelo y luego compararla en el espacio vectorial con los *embeddings* de los datos que se tienen calculados.

Tras identificar qué recursos son más afines a la consulta mediante la medición de la similitud coseno, se utilizan los índices creados en D3L para buscar otros recursos que sean similares a los encontrados. Con todos los recursos que se recuperaron, se crea el contexto que se le proveerá al LLM para que devuelva la respuesta al usuario. El *prompt* al LLM utilizado para la generación de texto se encuentra en el anexo C.5, donde se ve cómo se listan los recursos relacionados a la *query* del usuario y además los relacionados a estos con la similaridad de D3L, siendo estos mostrados como "Recursos Extra".

## 5.6. Mantenimiento del sistema

Ya presentada la herramienta final, es necesario especificar el proceso de mantenimiento del sistema. Aunque esto no se implementó en código por motivos de alcance del proyecto, se diseñó de forma detallada. A continuación, se especifica cuáles son las tareas que se deben realizar de forma que los mecanismos de búsqueda y exploración soporten la ingesta, actualización y borrado de conjuntos de datos.

### 5.6.1. Ingesta

Para poder actualizar la estructura subyacente del sistema en caso de que se incorpore un nuevo conjunto de datos, se deben tener las siguientes consideraciones en las diferentes etapas:

- **Preprocesamiento**: El nuevo *dataset* debe pasar por el proceso de verificación de calidad de datos para asegurar que su formato, *encoding* y tamaño sean consistentes.

- **D3L**: Se calcula el valor de similaridad del *dataset* para los cinco índices. Esto solo aplica para el nuevo conjunto de datos, observar que el resto mantendrá su valor de similaridad actual.

- **Aurum y RDF**: Debido a que se incorporó una nueva entrada en los LSH de D3L, también se deben actualizar las relaciones sintácticas del grafo de salida. No es necesario computar el grafo entero desde cero, ya que las relaciones que existían antes se mantendrán, solo se agregarán las nuevas correspondientes al nuevo *dataset*.

- **TableMiner+**: Únicamente para el conjunto de datos entrante, se deben ejecutar los procesos de detección de columna sujeto y clasificación de tipos de datos para todas las columnas. Posteriormente, se ejecuta TableMiner+ para las anotaciones semánticas.
- **LLM**: Solo en caso de que el nuevo conjunto de datos no cuente con metadatos descriptivos se debe ejecutar el generador de descripciones, en cualquier otro caso se mantiene la información inicial.
- **RAG**: Se codifican los recursos relacionados a dicho *dataset* para ser agregados al espacio de *embeddings* sobre el cual se realiza la búsqueda para las *queries* del RAG.

Por último se agrega toda esta información a la ya existente terminando con la ingesta del nuevo conjunto de datos.

### 5.6.2. Borrado

En caso de que se tenga que remover un conjunto de datos, se debe eliminar toda información relacionada al mismo, sin dejar rastro. A continuación se describe lo que se debe borrar:

- **D3L**: Se deben eliminar las entradas correspondientes al *dataset* en los LSH.
- **Aurum y RDF**: Se buscan todas las aristas y nodos relacionados al conjunto que se está eliminando, y se remueven.
- **RAG**: Se deben eliminar los *embeddings* correspondientes a los recursos relacionados, de forma que no se tengan en cuenta para futuras búsquedas del RAG.

Finalmente, se deben borrar todos los archivos relacionados al recurso, incluidos los creados por el sistema.

### 5.6.3. Actualización

En caso de que uno de los conjuntos de datos existentes se modifique, se procede eliminando toda la información relacionada con el mismo (incluida la generada por el sistema). Luego, solo para ese *dataset*, se vuelve a ejecutar el proceso de ingesta desde cero, como si se tratase de una nueva adquisición. En resumen, primero se procede tal y como se detalla en la sección anterior de borrado, y luego tal y como se especifica en la de ingesta.

# Capítulo 6

# Experimentación

En este capítulo se detallan los experimentos realizados para evaluar cada una de las herramientas utilizadas. Antes de presentar los experimentos, se introducen las métricas de evaluación, el entorno de ejecución y se muestra el proceso de selección de conjuntos de datos utilizados.

## 6.1. Métricas de evaluación

Las métricas de evaluación utilizadas fueron *precision*, *recall* y F1, las cuales se presentan a continuación.

### 6.1.1. Precisión

La *precision* evalúa la proporción de instancias clasificadas como positivas que realmente son positivas [23]. El objetivo es obtener un valor alto de *precision*, lo que refleja un bajo número de falsos positivos en las predicciones.

Se calcula como:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Donde:

- **TP** (Verdaderos Positivos): Instancias correctamente identificadas como positivas.
- **FP** (Falsos Positivos): Instancias incorrectamente clasificadas como positivas.

### 6.1.2. Recall

El *recall*, también conocido como sensibilidad o tasa de verdaderos positivos, mide cuántas de las instancias positivas del total fueron correctamente identificadas [23]. El objetivo es obtener un valor alto de *recall*, lo que indica que el modelo está capturando la mayoría de las instancias positivas.

Se calcula como:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Donde:

- **TP** (Verdaderos Positivos): Instancias positivas correctamente identificadas.
- **FN** (Falsos Negativos): Instancias positivas que no fueron identificadas.

### 6.1.3. F1

El puntaje F1 combina la *precision* y el *recall* en una sola métrica, calculando su media armónica [23]. El objetivo es maximizar el puntaje F1, lo que indica un buen equilibrio entre *precision* y *recall*, especialmente en escenarios con clases desbalanceadas.

Se calcula como:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

El puntaje F1 varía entre 0 y 1, donde 1 indica un equilibrio perfecto entre *precision* y *recall*.

## 6.2. Entorno de ejecución

Todas las pruebas que se realizaron se llevaron a cabo en un entorno de trabajo remoto, provisto por Google Collab Pro (versión paga) con las siguientes características:

| **Componente** | **Modelo** | **Capacidad** |
|---|---|---|
| RAM | DDR4 | 53.9 GB |
| GPU | NVIDIA L4 Tensor Core GPU | 22.5 GB VRAM |
| Disco | SSD | 235.7 GB |

Tabla 6.1: Características del entorno de ejecución en Colab Pro.

La elección de este entorno se debe a que algunos de los experimentos realizados tardaban alrededor de un día en ejecutar. Esto significaba una sobrecarga sobre los equipos personales y la memoria RAM no era suficiente, causando interrupciones en la ejecución de las pruebas. Utilizar un entorno de ejecución remoto, posibilitó acceso a mejores recursos de *hardware* y agilizó el proceso de experimentación. Por otro lado, en este entorno de ejecución se tenían a disposición varias GPU, lo cual era indispensable para poder ejecutar un LLM. En particular, el modelo de lenguaje *Meta-Llama-3.2-3B-Instruct* requiere cerca de 18 GB de VRAM solo para cargarse. Una vez en ejecución, los 4.5 GB restantes de VRAM en la GPU se van llenando conforme se utiliza el modelo, principalmente debido al almacenamiento en caché.

### 6.2.1. Datos utilizados

Como se detalla en 4.2.1, la mayoría de los conjuntos de datos en el catálogo consistían en tablas multidimensionales con mediciones, cuyas columnas en su mayoría eran numéricas. En particular, para la evaluación de TableMiner+, es necesario que las tablas tengan al menos una columna *named_entity*, por lo cual una vez finalizada la descarga y preprocesamiento de los datos detallada en

4.3, se realizó una selección manual de conjuntos de datos priorizando este tipo de tablas.

Como resultado del proceso de selección manual, se eligieron 63 conjuntos de datos representativos de distintos ámbitos del catálogo. Estos datos cuentan con 400 columnas en total, es decir, un promedio de 6.35 columnas por *dataset*. Para evaluar cómo la cantidad de filas con las que se trunca el *dataset* impacta en el desempeño de las herramientas, se generaron tres variantes de cada conjunto de datos, con 10, 20 y 50 filas en los recursos CSV respectivamente, lo que permitió analizar el efecto del tamaño de la muestra en los resultados.

## 6.3. TableMiner+

La evaluación del desempeño de TableMiner+ se llevó a cabo en dos etapas principales. Primero, se utilizó el método de evaluación del desafío SemTab [26], que proporciona un *ground truth* (GT) y un evaluador de código abierto. Esta fase permitió comparar la salida del sistema con una referencia validada. Finalmente en esta etapa, se compararon los resultados obtenidos con los del desafío en 2023 para los mismos datos utilizados.

En la segunda etapa, se evaluó TableMiner+ con datos del Catálogo de Datos Abiertos de Uruguay, para lo cual se construyó manualmente un GT específico para las tablas seleccionadas. Este GT no solo permitió evaluar la precisión de las anotaciones semánticas generadas por TableMiner+, sino también la clasificación de columnas en categorías como *number*, *named_entity*, *date_expression*, entre otras.

En ambas etapas, se calcularon las métricas de F1, *precision* y *recall* para las anotaciones semánticas.

### 6.3.1. Evaluación utilizando tablas de SemTab

Si bien el objetivo principal de este proyecto es aplicar las herramientas al Catálogo de Datos Abiertos del Gobierno, para realizar una primera evaluación y verificar el impacto de las mejoras implementadas en TableMiner+ (detalladas en 5.3.1), era fundamental disponer de un conjunto significativo de *datasets* previamente anotados, para comparar la salida del sistema con una referencia validada y evaluar su desempeño de manera objetiva.

Existen varios *benchmarks* ampliamente utilizados para la anotación semántica de tablas, cada uno con diferentes anotaciones de referencia [19]. El desafío anual *SemTab* [26] se ha convertido en un referente en la tarea de coincidencia de datos tabulares con bases de conocimiento, desarrollando *benchmarks* tanto reales como sintéticos de diversos tamaños y tareas [19], incluyendo:

- Anotación de celdas con entidades de una base de conocimiento (CEA).
- Anotación columnas con concepto (tipos semánticos) de una base de conocimiento (CTA).
- Anotación de relaciones entre columnas con propiedades de una base de conocimiento (CPA).

Estos *benchmarks* abarcan diferentes bases de conocimiento estructuradas, como DBpedia, Wikidata y Schema.org. En particular, el presente trabajo se centró únicamente en la tarea de anotación de columnas con conceptos semánticos (CTA) y se utilizó Wikidata como base de conocimiento.

La página oficial de SemTab incluye carpetas para cada edición anual, con los *datasets* utilizados para la evaluación en cada edición, el código de los evaluadores oficiales para cada tarea y los GT. Se utilizó el evaluador correspondiente a la tarea CTA, junto con los datos más recientes disponibles en SemTab (año 2023) [15] para evaluar TableMiner+.

Se decidió utilizar este enfoque para evaluar TableMiner+ no solo porque los datos y herramientas estaban disponibles públicamente, sino también porque SemTab es un desafío de referencia en el estado del arte de la anotación semántica de tablas.

### Métricas de evaluación

En la tarea CTA, se anota cada columna con un tipo semántico. Dado que Wikidata tiene una jerarquía de tipos muy detallada, es importante considerar ancestros y descendientes en la evaluación de la anotación, ya que en algunos casos, si bien el tipo semántico con el que se anota la columna no es exactamente el esperado, sigue siendo válido en un nivel más general. Por ejemplo, si la columna tiene como tipo semántico "mamífero", y el sistema la anota como "animal", no es tan incorrecto como si la anotara como "automóvil".

En la evaluación de SemTab, se utiliza una versión modificada de las métricas de *precision* y *recall* para esta tarea, donde se consideran las anotaciones parcialmente correctas, es decir, aquellas que sean ancestros o descendientes de las clases del GT [16].

El evaluador recibe un archivo CSV con las anotaciones devueltas por el sistema en el siguiente formato:

```
nombre_de_tabla, índice_de_columna, anotación
```

Para cada columna anotada, el sistema compara la anotación devuelta con el GT. Si no coinciden, se verifica si la entidad devuelta por TableMiner+ es un ancestro o descendiente de la entidad correcta en el KG.

La puntuación de corrección $cscore(\alpha)$ de una anotación $\alpha$ se calcula considerando su distancia a las clases del GT, definida como:

$$cscore(\alpha) = \begin{cases} 0{,}8^{d(\alpha)}, & \text{si } \alpha \text{ está en GT o es un ancestro con } d(\alpha) \leq 5 \\ 0{,}7^{d(\alpha)}, & \text{si } \alpha \text{ es un descendiente de GT con } d(\alpha) \leq 3 \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{6.1}$$

Donde $d(\alpha)$ representa la distancia más corta de $\alpha$ a una de las clases del GT.

A partir de $cscore(\alpha)$, se calculan las métricas ajustadas de *precision* (AP), *recall* (AR) y F1 (AF1) para CTA de la siguiente forma:

$$AP = \frac{\sum cscore(\alpha)}{|\text{Anotaciones del Sistema}|} \tag{6.2}$$

$$AR = \frac{\sum cscore(\alpha)}{|\text{Anotaciones Objetivo}|} \tag{6.3}$$

$$AF1 = \frac{2 \times AP \times AR}{AP + AR} \tag{6.4}$$

Donde |Anotaciones del Sistema| representa la cantidad de columnas que el detector de tipos de columna clasificó como *named_entity*.
Dado que, en el caso de TableMiner+, esta clasificación depende directamente del correcto funcionamiento del detector de tipos, en este trabajo se propusieron nuevas métricas denominadas *Real Precision* (RP) y *Real F1* (RF1). Estas métricas calculan la *precision* considerando únicamente las anotaciones del sistema que realmente debieron ser anotadas (es decir, aquellas que están presentes en el GT). Observar que *recall* no cambia porque el denominador son las anotaciones objetivos.

Las columnas clasificadas erróneamente como *named_entity* no se tienen en cuenta en esta métrica, ya que no existe una referencia válida en el GT para determinar si el concepto devuelto es correcto. En la *precision*, incluso si la anotación es correcta, estas clasificaciones incorrectas se consideran errores, afectando injustamente la evaluación del sistema de anotación por fallos que en realidad corresponden al detector de tipos.

#### Diseño del experimento

Para evaluar el impacto de las mejoras implementadas en TableMiner+, se diseñó un experimento en tres etapas que permitió analizar de manera aislada las modificaciones realizadas.

El objetivo era determinar si las mejoras incrementaban el rendimiento del sistema sin introducir efectos no deseados, y asegurarse de que la incorporación del mecanismo de *fallback* basado en LLM no enmascarara posibles fallas introducidas con los cambios realizados. El riesgo de introducir el mecanismo de *fallback* del LLM desde el principio, es que "corrija" automáticamente algunas anotaciones erróneas, enmascarando cualquier problema introducido con la implementación mejorada.

El proceso de evaluación se llevó a cabo en los siguientes tres pasos:

- **Paso 1:** Se ejecutó TableMiner+ en su versión base, sin ninguna de las mejoras introducidas en este trabajo. Únicamente se agregó la implementación de la caché (detallada en 5.3.1) y correcciones de errores que impedían la ejecución (ej. divisiones por cero en el cálculo de métricas).

  Esta primera ejecución permitió establecer una línea base de rendimiento, contra la cual se compararon posteriormente las mejoras.

- **Paso 2:** Se realiza una nueva ejecución pero incluyendo las mejoras desarrolladas en este trabajo (detalladas en 5.3.1), pero sin activar el mecanismo de *fallback* basado en LLM.

Esta segunda fase se realizó con el objetivo de evaluar el impacto real de las mejoras implementadas, asegurándose de que efectivamente mejoran el rendimiento del sistema y no generaban un descenso en la *precision* o *recall*.

- **Paso 3:** Se ejecuta nuevamente pero esta vez incluyendo el mecanismo de *fallback* (detallado en 5.3.2). En este paso se evalúa la mejora que representa introducir el LLM a TableMiner+ para los casos en que no termina obteniendo una anotación.

### Datos utilizados

Se seleccionaron 116 tablas del conjunto de datos WikidataTables utilizado en *SemTab* de 2023 [17]. Este conjunto de datos contiene tablas realistas generadas utilizando consultas SPARQL [16]. Estas tablas fueron creadas filtrando etiquetas que pueden referirse a más de una entidad en Wikidata, forzando un alto nivel de ambiguedad en las columnas.

Además, las tablas tienen ruido introducido a propósito, lo que también permite evaluar cómo se recupera el sistema ante errores ortográficos o de tipeo. La tabla 6.2 presente en los datos, es un ejemplo de esto. Los datos correctos deberían ser Vermilion Parish, Red River Parish, Bossier Parish, Union Parish, Livingston Parish, y West Feliciana Parish.

| **col0** |
|---|
| Vermilipn Parish |
| Re River Parish |
| Bossier Pbrish |
| Uniop Parish |
| iwingston Parish |
| est Felician Parish |

Tabla 6.2: Columna con ruido presente en los datos de SemTab 2023

### Resultados obtenidos

La etapa de la detección del tipo de las columnas detectó 123 columnas *named_entity*, de las cuales 23 no están en el GT (fueron clasificadas incorrectamente como *named_entity*).

Los resultados obtenidos para los tres pasos se detallan en la tabla 6.3, donde se muestra, para cada paso:

- La cantidad de columnas que fueron clasificadas con el concepto exactamente correcto del GT.
- La cantidad de columnas clasificadas con un ancestro o descendiente del concepto del GT.
- Las métricas detalladas anteriormente.

En la tabla 6.4 se muestran la *real f1*, *real precision* y *recall* (esta última se mantiene igual) obtenidas teniendo en cuenta las columnas clasificadas correctamente como *named_entity*.

| Paso | Exactas | Ancestro/Desc. | F1 | Precision | Recall |
|---|---|---|---|---|---|
| 1 | 24 | 14 | 0.256 | 0.279 | 0.237 |
| 2 | 41 | 23 | 0.428 | 0.466 | 0.395 |
| 3 | 43 | 26 | 0.459 | 0.500 | 0.424 |

Tabla 6.3: Resultados de clasificación por pasos

| Paso | Real F1 | Real precision | Recall |
|---|---|---|---|
| 1 | 0.281 | 0.344 | 0.237 |
| 2 | 0.468 | 0.573 | 0.395 |
| 3 | 0.502 | 0.615 | 0.424 |

Tabla 6.4: Resultados sin tener en cuenta columnas detectadas erróneamente como *named_entity*

Los resultados muestran una mejora progresiva en la clasificación tras cada paso, lo que evidencia la efectividad de las modificaciones implementadas en la herramienta.

Además, al aumentar la cantidad de filas en las tablas, se observa una mejora en los resultados. Esto sugiere que una mayor cantidad de datos disponibles permite realizar inferencias más precisas.

Por otro lado, la falla del detector de tipos al clasificar columnas incorrectamente como *named_entity* introduce un error fijo, ya que en este caso 23 columnas anotadas siempre contenían errores. Sin embargo, al calcular *real precisión* y *real f1*, dividiendo entre las columnas correctamente clasificadas, se puede evaluar el rendimiento de la herramienta sin la influencia del detector de tipos.

### Impacto del ruido en los datos

Tras una revisión manual de las 31 columnas clasificadas erróneamente en la salida del paso 3, se identificó que 12 de ellas contenían ruido introducido intencionalmente, similar al caso ilustrado en la Figura 6.2.

Al corregir manualmente los errores tipográficos y ortográficos en los datos de SemTab, los resultados mejoraron significativamente, como se muestra en la Tabla 6.5.

| Exactas | Ancestro/Desc. | F1 | Precision | Recall |
|---|---|---|---|---|
| 47 | 31 | 0.515 | 0.561 | 0.476 |

Tabla 6.5: Resultados de clasificación sin ruido

En la tabla 6.6 se muestran las métricas “reales” para este caso.

Los resultados indican que muchas de las anotaciones erróneas en el conjunto de datos original se debían a errores tipográficos y ortográficos presentes en datos de SemTab. Esto demuestra que el sistema no maneja adecuadamente la presencia de ruido en los datos, lo cual representa una limitación importante, dado que es posible que los datos en el Catálogo de Datos Abiertos de Uruguay contengan errores similares.

| Real F1 | Real precision | Recall |
|---|---|---|
| 0.563 | 0.689 | 0.476 |

Tabla 6.6: Resultados sin tener en cuenta columnas detectadas erróneamente como *named_entity*

#### Comparación con resultados de SemTab 2023

La tabla 6.7 muestra la *precision* para los 5 sistemas participantes en SemTab 2023 en la tarea CTA con los datos WikidataTables.

| Task | TSOTSA | Kepler-aSI | TorchicTab | MUT2KG | Semtex |
|---|---|---|---|---|---|
| CTA | 0.738 | 0.739 | 0.749 | 0.655 | 0.934 |

Tabla 6.7: Resultados de CTA en SemTab 2023 [16]

En 2023 en particular, no hubo un sistema que se destacara en la mayoría o en todas las tareas, y los resultados obtenidos fueron, en general, inferiores a los de años anteriores [16]. Esta disminución en el desempeño puede deberse a un mayor nivel de dificultad en los datos actuales. En el presente trabajo, se obtuvieron resultados de *precision* y *real precision* de 0.561 y 0.689 respectivamente. Esto sitúa a la implementación del presente trabajo de TableMiner+ en una posición cercana a MUT2KG.

### 6.3.2. Evaluación utilizando tablas del Catálogo de Datos Abiertos de Gobierno

Una vez evaluadas las mejoras realizadas a TableMiner+ con los datos de *SemTab*, se procedió a evaluar su funcionamiento en el contexto del caso de estudio del catálogo.

#### Construcción de GT

En primer lugar, como para los datos del catálogo no se contaba inicialmente con un GT contra el cual se pudiesen medir los resultados, se realizaron anotaciones manuales sobre los conjuntos de datos seleccionados, recorriendo cada columna de las tablas y anotando manualmente su tipo de datos. Luego de estas anotaciones, se extrajeron métricas indicando cuantos datos pertenecían a cada una de las categoría, obteniendo los siguientes resultados:

- 154 columnas del tipo *number*
- 141 columnas del tipo *named_entity*
- 60 columnas del tipo *date_expression*
- 4 columnas del tipo *long_text*
- 1 columnas del tipo *empty*
- 40 columnas del tipo *other*

Posteriormente, para las columnas clasificadas como *named_entity*, se buscó un concepto asociado en Wikidata.

Se creó un CSV con el siguiente formato:

```
nombre_de_tabla, índice_de_columna, tipo, anotación
```

Para poder obtener las métricas, se adaptó el evaluador utilizado en la sección 6.3.1 para que utilice el GT generado a partir de las anotaciones manuales. Además, se implementó un nuevo evaluador para medir la precisión del detector de tipos de columnas, ya que esto influye directamente en los resultados de TableMiner+.

Al igual que en la evaluación con tablas de *SemTab*, se calcula F1, *precision*, *recall* y las correspondientes métricas "reales" (utilizando solo las columnas anotadas que efectivamente son *named_entity*).

### Diseño del experimento

Se ejecutó TableMiner+ con los 63 *datasets* seleccionados (detallado en la sección 6.2.1), variando la cantidad de filas de los *datasets* descargados y el *prompt* utilizado para encontrar el concepto en el mecanismo de *fallback*.

Estos experimentos también se realizaron en etapas: en primer lugar se ejecutó TableMiner+ para un *prompt* específico, variando la cantidad de filas de las tablas entre 10, 20 y 50 y midiendo para cada una el tiempo de ejecución. Luego se probó un segundo *prompt*, y se repitió el procedimiento.

### Resultados obtenidos

En primer lugar, se evaluó el detector de tipos para cada cantidad de filas. En la tabla 6.8 se muestra para cada cantidad de filas, la cantidad de columnas que fueron clasificadas y la cantidad de columnas correcta e incorrectamente clasificadas.

| filas | Columnas | Correctas | Incorrectas |
|---|---|---|---|
| 10 | 400 | 353 | 47 |
| 20 | 400 | 356 | 44 |
| 50 | 400 | 355 | 45 |

Tabla 6.8: Resultados del detector de tipo de columna

De los errores, la mayoría son confusiones entre columnas clasificadas como *other* o *long_text* en el GT que el detector de tipos clasifica como *named_entity*. Otros casos que causan error son columnas clasificadas como *empty* por el detector de tipos y en el GT están clasificadas con un tipo específico. Esto puede deberse a la proporción de celdas vacías que se tienen en cuenta para considerar una columna como *empty*.

Según el GT construido manualmente, de las 400 columnas, 141 son *named_entity*. En la tabla 6.9 se muestra para cada cantidad de filas, la cantidad de columnas clasificadas como *named_entity* y cuántas de estas son correctas.

| filas | named_entity | correct named_entity | incorrect named_entity |
|---|---|---|---|
| 10 | 155 | 129 | 26 |
| 20 | 158 | 134 | 24 |
| 50 | 155 | 132 | 23 |

Tabla 6.9: Resultados de clasificación en named_entity

Se observa que el detector de tipos comete errores al clasificar varias columnas, lo que genera un error fijo en las anotaciones posteriores de TableMiner+. Esto sugiere que mejorar su precisión podría reducir el impacto de estos errores en las siguientes etapas del proceso.

En este caso, para calcular *precision*, el denominador será el de la columna *named_entity* de la tabla 6.9 y para calcular *real precision*, el denominador será el de la columna *correct named_entity*.

En la tabla 6.10 se muestran los resultados obtenidos al probar con el primer *prompt* (detallado en el anexo C.4.1), que utiliza toda la información que se tiene acerca del paquete que contiene la tabla (tabla, metadata completa, notas, descripción del paquete).

| filas | tiempo (min) | Correctas | Incorrectas | F1 | Precision | Recall |
|---|---|---|---|---|---|---|
| 10 | 26.5 | 39 | 92 | 0.264 | 0.252 | 0.277 |
| 20 | 35.9 | 43 | 91 | 0.288 | 0.272 | 0.305 |
| 50 | 67.4 | 50 | 82 | 0.338 | 0.323 | 0.355 |

Tabla 6.10: Resultados de TablerMiner+ con prompt 1 (ver anexo C.4.1)

En la tabla 6.11 se muestran las métricas "reales" para este caso:

| Filas | Real F1 | Real precision | Recall |
|---|---|---|---|
| 10 | 0.289 | 0.302 | 0.277 |
| 20 | 0.313 | 0.321 | 0.305 |
| 50 | 0.366 | 0.379 | 0.355 |

Tabla 6.11: Resultados sin tener en cuenta columnas detectadas erróneamente como *named_entity*

En la tabla 6.12 se muestran los resultados obtenidos al probar con el segundo *prompt* (detallado en el anexo C.4.2), que omite el contexto y se utiliza simplemente los valores únicos de la columna, el título de la tabla y el nombre de la columna.

En la tabla 6.13 se muestran las métricas "reales" para este caso:
Tras la ejecución para cada escenario, la salida fue evaluada manualmente para identificar posibles conceptos correctos que podrían haber sido descartados por no coincidir exactamente con el GT. El GT se fue actualizando y en algunos casos se agregaron varias opciones correctas para una anotación.

Los resultados muestran una caída significativa en *precision*, *recall* y F1 al probar con los datos

| filas | tiempo (min) | Correctas | Incorrectas | F1 | Precision | Recall |
|---|---|---|---|---|---|---|
| 10 | 25.3 | 42 | 89 | 0.284 | 0.271 | 0.298 |
| 20 | 38.9 | 45 | 89 | 0.301 | 0.285 | 0.319 |
| 50 | 68.0 | 51 | 81 | 0.345 | 0.329 | 0.362 |

Tabla 6.12: Resultados de TableMiner+ con prompt 2 (ver anexo C.4.2)

| Filas | Real F1 | Real precision | Recall |
|---|---|---|---|
| 10 | 0.309 | 0.321 | 0.298 |
| 20 | 0.327 | 0.336 | 0.319 |
| 50 | 0.374 | 0.386 | 0.362 |

Tabla 6.13: Resultados sin tener en cuenta columnas detectadas erróneamente como *named_entity*

del catálogo, con respecto a los resultados obtenidos en las pruebas con los datos de SemTab en la sección 6.3.1.

El uso de ontologías externas aporta nueva evidencia a las tareas de descubrimiento y exploración de datos, pero su efectividad depende de la cobertura de la ontología. Si la ontología utilizada no refleja correctamente el dominio de los datos, los resultados pueden ser imprecisos [20].

En este caso, los datos abiertos de Uruguay están en español y pertenecen a un dominio local específico, lo que dificulta encontrar un mapeo adecuado en Wikidata, que tiene un sesgo hacia entidades más globales. A pesar de la selección manual de tablas con al menos una columna clasificable como *named_entity*, la cobertura de la ontología afectó los resultados obtenidos.

Finalmente, los experimentos muestran que el segundo *prompt* supera al primero en todas las métricas. Esto sugiere que un exceso de contexto puede confundir al modelo, mientras que un *prompt* más simple y enfocado en la columna a clasificar, mejora el rendimiento.

## 6.4. Generación de metadatos

En esta etapa se pretende evaluar la calidad de las descripciones generadas por el LLM a nivel de tabla y de columnas. Para esto, era necesario contar con descripciones de referencia contra las cuales comparar. Se seleccionaron 50 *datasets* de los 63 que se utilizaron anteriormente, priorizando aquellos cuyo contexto fuese lo suficientemente descriptivo para poder brindarle información al LLM.
Para estos *datasets* se realizó un análisis manual, y se agregaron/modificaron descripciones en casos donde las provistas por el catálogo no fueran lo suficientemente descriptivas.

Ya que la estrategia de generación de metadatos varía según la información con la que se cuente (como se ve en 5.4), para comenzar la evaluación, los conjuntos de datos fueron divididos en tres grupos iguales con el siguiente tratamiento:

- Grupo 1: Se eliminó el archivo de metadatos, conservando únicamente la descripción general.

- Grupo 2: Se eliminó la descripción general, dejando únicamente el archivo de metadatos.
- Grupo 3: Se eliminaron tanto el archivo de metadatos como la descripción general.

La información eliminada fue almacenada para su posterior comparación, funcionando como la referencia (GT) contra la cual se evaluó la calidad de los metadatos generados por el modelo.

### 6.4.1. Métricas de Evaluación

Para evaluar la calidad de las descripciones generadas, se utilizó la métrica BERTScore[1] que permite evaluar la similitud semántica entre dos textos. Esta métrica se calculó de manera independiente para las descripciones generales de cada *dataset* y para las descripciones individuales para cada columna (contenidas en los archivos de metadatos) donde se evaluó la similitud entre la descripción generada por el LLM y la descripción original presente en el conjunto de datos, calculando así las métricas de *precision*, *recall* y F1.

### 6.4.2. Variantes de Ejecución

Para estudiar el impacto de diferentes configuraciones en la calidad de los resultados, se llevaron a cabo múltiples ejecuciones del modelo, variando los siguientes hiperparámetros:

- Temperatura: Se ajustó la temperatura del modelo para controlar el nivel de aleatoriedad en las respuestas generadas, probando valores de 0.3, 0.65 y 1.0.
- Top-p: Se aplicaron técnicas de muestreo por núcleo con valores de 0.8 y 0.9 para limitar la probabilidad acumulada de selección de palabras.
- Repetition Penalty: Se ajustó el penalizador de repetición para evitar descripciones redundantes, con valores de 1.0 (sin penalización) y 1.2.

Se realizaron estas variaciones de hiperparámetros utilizando los *datasets* truncados en 20 filas. Luego, con la mejor combinación de hiperparámetros según los resultados obtenidos, se probó con 10 y 50 filas en los CSV. La cantidad de filas no se tomó como uno de los hiperparámetros para determinar al inicio, pues el tiempo de ejecución podría haber escalado hasta el triple[2].

Cabe destacar que ciertas variables como el tipo de modelo, instrucciones en los *prompts* y los *few-shots* no fueron variados como parte de esta fase de ajuste al modelo, debido a que se llevó a cabo un esfuerzo de experimentación manual para su elección donde se pudo confirmar que el margen de mejora era muy reducido con los recursos disponibles y variaciones en los mismos generaban mínimas diferencias en los resultados.

[1] Mide la similitud entre textos predichos (candidatos) y textos de referencia comparando sus *embeddings* contextuales utilizando un modelo de lenguaje previamente entrenado.

[2] Para cada combinación de hiperparámetros, hubiese sido necesario probar con 10, 20 y 50 filas del CSV, dando como resultado el triple de combinaciones posibles.

### 6.4.3. Resultados de la evaluación

En total fueron una suma de 12 configuraciones distintas a probar, las cuales se corrieron tres veces cada una y se hizo el promedio de los resultados con el fin de reducir el sesgo que impondría el evaluar una única muestra para cada combinación de hiperparámetros. La ejecución tomó aproximadamente 24 horas, y los resultados se pueden ver en la tabla 6.14. Se puede observar que no hay mucha variación, pero la combinación de hiperparámetros que mejor luce según las métricas resultantes es la que posee valores 0.3, 0.8 y 1.2 de temperatura, *top-p* y *repetition penalty* respectivamente.

| Hiperparámetros | | | BERT-Score (dataset) | | | BERT-Score (columns) | | |
|---|---|---|---|---|---|---|---|---|
| Temperatura | Top p | Repetition Penalty | Precision | Recall | F1 | Precision | Recall | F1 |
| 0.3 | 0.8 | 1.0 | 0.666 | 0.777 | 0.714 | 0.723 | 0.790 | 0.754 |
| 0.3 | 0.8 | 1.2 | 0.707 | 0.778 | 0.739 | 0.714 | 0.768 | 0.739 |
| 0.3 | 0.9 | 1.0 | 0.660 | 0.770 | 0.708 | 0.720 | 0.786 | 0.750 |
| 0.3 | 0.9 | 1.2 | 0.708 | 0.786 | 0.743 | 0.709 | 0.762 | 0.734 |
| 0.65 | 0.8 | 1.0 | 0.662 | 0.782 | 0.714 | 0.709 | 0.786 | 0.744 |
| 0.65 | 0.8 | 1.2 | 0.701 | 0.780 | 0.737 | 0.702 | 0.757 | 0.727 |
| 0.65 | 0.9 | 1.0 | 0.674 | 0.780 | 0.721 | 0.695 | 0.776 | 0.732 |
| 0.65 | 0.9 | 1.2 | 0.689 | 0.774 | 0.729 | 0.689 | 0.754 | 0.719 |
| 1.0 | 0.8 | 1.0 | 0.691 | 0.789 | 0.735 | 0.665 | 0.762 | 0.709 |
| 1.0 | 0.8 | 1.2 | 0.659 | 0.758 | 0.703 | 0.651 | 0.729 | 0.687 |
| 1.0 | 0.9 | 1.0 | 0.655 | 0.776 | 0.707 | 0.660 | 0.750 | 0.701 |
| 1.0 | 0.9 | 1.2 | 0.648 | 0.735 | 0.687 | 0.648 | 0.723 | 0.683 |

Tabla 6.14: Evaluación del LLM para generación de descripciones.

Utilizando la mejor combinación de hiperparámetros, se varió la cantidad de filas de los CSV entre 10, 20 y 50. Viendo la tabla 6.15 con los resultados, se puede observar que 20 filas como contexto al *prompt* parece ser la alternativa más eficiente para el caso de estudio, aunque la diferencia no es muy grande, podría concluirse que 50 filas de ejemplo en el *prompt* puede ser un poco contraproducente ya que puede terminar por confundir al modelo, y esa puede ser la razón por la cual obtuvo los peores resultados. En el otro lado, tal vez 10 filas de ejemplo en el *prompt* no sea suficiente contexto en algunos casos.

| Hiperparámetros | BERT-Score (package) | | | BERT-Score (columns) | | |
|---|---|---|---|---|---|---|
| Cantidad de filas del CSV | Precision | Recall | F1 | Precision | Recall | F1 |
| 10 | 0.701 | 0.789 | 0.741 | 0.714 | 0.761 | 0.736 |
| 20 | 0.707 | 0.778 | 0.739 | 0.714 | 0.768 | 0.739 |
| 40 | 0.699 | 0.777 | 0.734 | 0.708 | 0.757 | 0.731 |

Tabla 6.15: Evaluación del LLM para generación de descripciones.

Finalmente, analizando las métricas resultantes del mejor resultado obtenido en la generación

de descripciones con el LLM, se ve que en el caso de las descripciones generales a nivel de *dataset* se obtuvo una *precision* de 0.707 y un *recall* de 0.778 lo que es un indicador de que el modelo efectivamente generó descripciones que engloban en su gran mayoría a las expresiones y términos que se utilizaron en la descripción original, conteniendo un significado en conjunto similar. Se ve un buen balance entre estas dos métricas, lo cual significa que el modelo logra una buena combinación entre mantener la precisión del contenido sin comprometer al significado de la descripción, dando como resultado una medida f1 de 0.739. Observar que este mismo resultado se obtuvo para las descripciones por columna.

## 6.5. RAG

Para evaluar el desempeño del RAG en la tarea de recuperación de información para responder una determinada consulta, se diseñó un experimento centrado en verificar la capacidad del sistema para identificar los recursos requeridos en función de cada pregunta. Dado que la calidad de la respuesta en lenguaje natural depende directamente de la relevancia de los recursos recuperados, se decidió enfocar la evaluación en medir qué tan bien el sistema encontraba información útil, sin considerar cómo la expresaba.
En este experimento se reutiliza la misma selección de 50 *datasets* que se utilizó en la sección anterior.

En primer lugar, se definió un GT compuesto por 30 consultas, que incluían tanto un estilo formal como expresiones más coloquiales, con el objetivo de poner a prueba la versatilidad del RAG ante diferentes formas de plantear las preguntas. Además, para cada consulta se establecieron qué *datasets* son considerados útiles para ofrecer una respuesta adecuada.

Con esta base, se llevó a cabo el experimento ingresando las 30 consultas en el RAG y registrando los *datasets* que el sistema señalaba como relevantes. A continuación, se contrastaron esas respuestas con las definidas en el GT, considerándose satisfactoria la recuperación cuando al menos uno de los *datasets* propuestos coincidiera con la lista del GT de cada consulta. Como resultado, se observó que en 21 de las 30 consultas (70 % de los casos) el RAG localizó correctamente los recursos esperados, mientras que en las 9 restantes no se identificaron los datos esperados.

Si bien los resultados muestran que el RAG encuentra información útil en un número significativo de casos, es importante destacar que se trabajó con un entorno de pruebas reducido y no se disponía de muchos conjuntos de datos con solapamiento temático. En consecuencia, en la mayoría de las consultas solamente existía un único *dataset* que aportaba información relevante.

## 6.6. D3L y Aurum

Si bien D3L construye cinco índices LSH (ver sección 3.1), en el presente trabajo solo se utilizan los índices de nombre y valor para construir el grafo de Aurum que infiere relaciones sintácticas entre columnas de diferentes *datasets*. Por esta razón, la evaluación de D3L se limitó a mediciones del tiempo de construcción de los índices LSH variando la cantidad de filas en las que se truncan los *datasets*. Luego, en particular la evaluación de los índices de nombre y valor se realizó en conjunto

con la evaluación de Aurum como se describe en la siguiente sección.

### 6.6.1. Diseño del experimento

Para esta evaluación se utilizaron los 63 *datasets* seleccionados manualmente (ver sección 6.2.1). Como las relaciones que infiere Aurum son sintácticas entre los nombres y los valores de los atributos, para evaluar la calidad de estas relaciones primero fue necesario analizar manualmente los *datasets* para identificar columnas relacionadas según estos criterios, es decir que tuviesen nombre o valores de los atributos en común. Se identificaron columnas a analizar (por ejemplo "departamento") teniendo en cuenta las variaciones en los nombres de las columnas (por ejemplo "depto", "dpto", etc) y se contaron las ocurrencias en total en los 63 *datasets*. En la tabla 6.16 se muestran las columnas seleccionadas (en su representación general), las variaciones de nombres y la cantidad de ocurrencias en total en los 63 *datasets*.

Luego se construyó el grafo de Aurum, donde cada nodo representa una columna y las aristas representan relaciones entre columnas. Para cada grupo de columnas identificado (departamento, año, valor, etc), se analizó el subgrafo formado por sus columnas, asumiendo que todas ellas representan lo mismo. Bajo esta hipóstesis, estos subgrafos deberían ser totalmente conexos, es decir, todos los nodos de estos subgrafos deberían estar relacionados entre sí. Si esto se cumple, la cantidad de aristas de un subgrafo debería ser $\frac{n(n-1)}{2}$, siendo $n$ la cantidad de nodos. Se calculó para cada grupo de columnas, la cantidad de aristas que contiene su subgrafo con el fin de analizar que tan "conexo" resultó el subgrafo.

Además, se construyó el grafo en primer lugar con el umbral de similaridad[3] en 0.5 y luego se cambió a 0.3 para que tuviese más "tolerancia" a diferencias. Se registraron los cambios en las aristas al cambiar el umbral.

### 6.6.2. Resultados obtenidos

En la tabla 6.17 se muestra para cada grupo, la cantidad de columnas (nodos en el grafo), la cantidad de relaciones inferidas (aristas en el grafo) con ambos umbrales utilizados y la cantidad de aristas de un grafo totalmente conexo para esa cantidad de nodos.

En general los grupos de columnas analizados estaban relacionados en el grafo, tal como se esperaba. En particular, las columnas que representaban año (34 columnas) y valor (43 columnas), todas estaban unidas en un componente del grafo. Esto tiene sentido ya que los nombres son idénticos excepto por mayúsculas.

Las dos columnas "Mes" estaban relacionadas, seguramente también por tener nombres idénticos. Sin embargo, la columna "Meses", si bien en sus valores contenía también meses, no estaba relacionada con ninguna. Esto puede deberse a que entre los valores, si bien todos eran meses, la mayoría eran distintos a los valores de las otras dos columnas, y al utilizar similaridad sintáctica, esto puede influir en que directamente no detecte relación. Para probar esto, se introdujeron manualmente los mismos meses en la columna "Meses" y en este caso sí se detectó relación. Esto

[3]Valor mínimo del promedio entre similitud de nombre y de valor.

| Columna | Variaciones de nombres | Ocurrencias | Ocurrencias totales |
|---|---|---|---|
| valor | valor | 41 | 43 |
| | Valor | 2 | |
| año | año | 32 | 34 |
| | Año | 1 | |
| | AÑO | 1 | |
| mes | Mes | 2 | 3 |
| | Meses | 1 | |
| país | pais_prov | 1 | 7 |
| | Pais | 1 | |
| | PAIS | 3 | |
| | Países | 1 | |
| | Destino | 1 | |
| departamento | departamento_prov | 1 | 16 |
| | DEPARTAMENTO | 1 | |
| | departamento | 4 | |
| | Departamento_residencia | 1 | |
| | Departamento | 5 | |
| | Otro Departamento | 1 | |
| | departamento_residencia | 1 | |
| | IdDepartamentoDestino | 1 | |
| | IdOtroDepartamento | 1 | |
| sexo | sexo | 8 | 14 |
| | Genero | 6 | |
| nombre | nombre | 3 | 15 |
| | Nombre | 3 | |
| | NOMBRE_AP | 1 | |
| | NOMBRE_INC | 1 | |
| | NOMBRE_UE | 1 | |
| | NOMBRE_PROG | 1 | |
| | NOMBRE_PROY | 1 | |
| | ue_nombre | 1 | |
| | ap_nombre | 1 | |
| | programa_nombre | 1 | |
| | tipo_gasto_nombre | 1 | |

Tabla 6.16: Grupos de columnas seleccionados y sus ocurrencias

muestra la necesidad de una herramienta que incluya similaridad semántica como TableMiner+.

En las columnas que representan países, las siete ocurrencias totales tienen en sus valores países, pero sin embargo, todas las columnas quedan en un mismo componente del grafo, excepto las columnas “Destino”, “Países” y “pais_prov” que quedan sin relación, seguramente porque difieren sus valores y además el nombre no es idéntico. Para el umbral de 0.3, la columna “Destino”, si bien difiere totalmente en el nombre de la columna, queda igualmente relacionada con una de las

| Grupo | #Columnas | #Relaciones inferidas | | #Relaciones totales |
|---|---|---|---|---|
| | | (u=0.3) | (u=0.5) | |
| Valor | 43 | 903 | 903 | 903 |
| Año | 34 | 561 | 561 | 561 |
| Mes | 3 | 1 | 1 | 3 |
| País | 7 | 7 | 3 | 21 |
| Departamento | 16 | 16 | 80 | 120 |
| Sexo | 14 | 79 | 43 | 91 |
| Nombre | 15 | 22 | 1 | 102 |

Tabla 6.17: Resultados de aristas inferidas por cada subgrafo

columnas ya que tiene países similares entre sus valores.

Las columnas que representan sexo y género tienen en general los mismos valores (varones, mujeres, femenino, masculino, otro). Sin embargo, utilizando el umbral de 0.5 quedan dos componentes distintos: uno con las columnas "sexo" y otros con las columnas "género". Al bajar el umbral a 0.3, las 14 columnas quedan en un mismo componente conexo del grafo.
Un caso similar son las columnas que se refieren a nombres. Con el umbral en 0.5 todas las columnas quedan separadas excepto dos de las columnas "nombre" que ademas de coincidir en los nombres, ambas trataban de nombres de productos escolares, por lo que también coincidían en sus valores. El resto de columnas si bien su nombre es muy similar, quedaban separadas ya que sus valores diferían totalmente. Esto tiene sentido ya que "nombre" puede ser de productos, de ciudades, de autos, etc. Sin embargo, al bajar el umbral a 0.3, queda un componente del grafo que relaciona todas las columnas "nombre", "Nombre", "ue_nombre" y "ap_nombre", luego "NOMBRE_PROG" y "NOMBRE_PROY" quedan relacionadas y el resto de columnas quedan sin relación.

Finalmente, las columnas que contienen departamentos quedan todas relacionadas en un componente, excepto "IdDepartamentoDestino" que queda relacionada a columnas con nombre "Destino" o similares, y "Departamento_residencia" y "departamento_residencia" que quedan relacionadas pero separadas de las demás.

## 6.7. Tiempo de construcción de índices LSH

Para evaluar el desempeño de D3L en la generación de índices, se midió el tiempo de construcción de los índices, considerando distintos tamaños de truncamiento en los *datasets* procesados.

En la tabla 6.18 se muestran los tiempos requeridos para construir cada índice según cada tamaño de truncamiento. Se comenzó probando con truncamientos de 10, 20 y 50 filas, pero al notar que no había una variación significante en el tiempo de indexación, se volvió a ejecutar con un truncamiento de 500 filas.

Los resultados indicaron que el tiempo de indexación se mantuvo prácticamente constante en todas las configuraciones evaluadas, incluso con el truncamiento en 500 filas. Esto puede deberse a que

| Cantidad de filas | Name Index (s) | Format Index (s) | Value Index (s) | Distribution Index (s) | Embedding Index (s) |
|---|---|---|---|---|---|
| 10 | 0.211 | 0.201 | 0.731 | 0.592 | 113.407 |
| 20 | 0.214 | 0.233 | 0.891 | 0.625 | 115.266 |
| 50 | 0.225 | 0.329 | 0.930 | 0.607 | 114.170 |
| 500 | 0.248 | 1.358 | 2.032 | 0.647 | 114.475 |

Tabla 6.18: Tiempos de ejecución en segundos de los índices generados por D3L para diferentes niveles de truncamiento.

utilizar este truncamiento no asegura que todas las tablas efectivamente tengan esa cantidad de filas.

El segundo experimento consistió en la construcción del grafo de Aurum para los truncamientos de 10, 20 y 50 filas, registrando la cantidad total de aristas en cada caso. Los resultados se muestran en la tabla 6.19, donde se puede observar un aumento de la cantidad de aristas del grafo a medida que se aumenta la cantidad de filas de las tablas.

| Cantidad de filas | Aristas detectadas |
|---|---|
| 10 | 350 |
| 20 | 359 |
| 50 | 376 |

Tabla 6.19: Cantidad de aristas detectadas en los grafos considerando todas las columnas para distintos niveles de truncamiento.

# Capítulo 7

# Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del trabajo y las posibles mejoras a futuro.

## 7.1. Conclusiones

Con el fin de facilitar las tareas de descubrimiento y exploración de datos en *data lakes*, se plantearon cuatro objetivos. El primero fue realizar un estudio del estado del arte de las herramientas existentes para abordar el problema. Como resultado, se encontraron muchas propuestas con diferentes enfoques, mostrando que es un área de investigación en constante evolución. En particular, se encontró un repositorio con implementaciones simplificadas de herramientas para identificar relaciones sintácticas y semánticas entre los datos, y se utilizó como solución base.

Estas herramientas fueron adaptadas para su aplicación en el caso de estudio del Catálogo de Datos Abiertos de Uruguay. Para ello, cumpliendo el segundo objetivo del trabajo, se realizó un estudio de la calidad de datos del catálogo a través de la extracción de medidas. A partir de este análisis se identificaron oportunidades de mejora para el catálogo y esto dio lugar a la incorporación de nuevas herramientas a la solución base.

El tercer objetivo consistía en implementar un sistema que permita descubrir y explorar relaciones dentro de los datos del catálogo. Para esto, se integraron las herramientas encontradas junto con técnicas de inteligencia artificial, para abordar uno de los problemas principales del catálogo: la calidad de sus metadatos. Como resultado, se propuso el grafo RDF y el RAG como herramientas interactivas que permiten la navegación de datos y descubrir conjuntos de datos relevantes a partir de un requerimiento específico.

Se realizaron experimentos para evaluar todas las herramientas y se obtuvieron resultados satisfactorios, cumpliendo con el cuarto objetivo del proyecto. La herramienta final propuesta demostró cumplir con los objetivos planteados, pudiéndose considerar un prototipo para su futura integración con el catálogo.

## 7.2. Trabajo a Futuro

Aunque los resultados obtenidos fueron satisfactorios, el alcance de este proyecto no bastó para contemplar de forma exhaustiva todas las posibles mejoras. A continuación se indican posibles vías de continuación de este sistema, y las ventajas que provee cada una de las alternativas.

### 7.2.1. Análisis e incorporación de Starmie

Como se detalla en 5.2.1, con el tiempo que se habría invertido en el análisis de Starmie, se decidió priorizar otros objetivos que surgieron a lo largo del proyecto tras identificar los problemas de calidad del catálogo, como la generación de metadatos y la integración con el LLM para implementar el mecanismo de consulta en lenguaje natural. Sin embargo, la utilidad de la herramienta aún resulta aplicable al catálogo, ya que hay muchas tablas similares pero de distintos años u organizaciones publicadoras, que se podrían unir y enriquecer la información.

### 7.2.2. Vocabulario para el grafo RDF

El grafo RDF presentado en 5.5.1 incluye triplas cuyos predicados no siguen un vocabulario de un estándar oficial. Aunque esto sea suficiente para los objetivos inmediatos del proyecto, si se pretende escalar el sistema y ofrecer a los usuarios la posibilidad de realizar consultas en SPARQL, además de facilitar la interoperabilidad con otros sistemas similares, este enfoque no resultaría viable.

### 7.2.3. Copia del KG en ambiente local

En TableMiner+ 3.3 se detectó que el mayor consumo de tiempo provenía de las consultas al grafo de conocimiento remoto. Para remediar esto, se propuso integrar Blazegraph y Elasticsearch, tal como se describe en 5.3.1, con el fin de mantener una copia de la base de conocimientos en el entorno local y así evitar la latencia de las consultas de red. Aunque no se logró concretar esta mejora debido a limitaciones del entorno, sigue siendo una oportunidad de optimización importante.

### 7.2.4. Similitud semántica por ancestros y descendientes

Tanto en Aurum 3.2 como en el grafo RDF provisto 5.5.1, actualmente se puede buscar columnas que compartan la misma entidad de Wikidata. Sin embargo, este enfoque puede resultar algo rígido. Una posible mejora consiste en incorporar métodos de similitud más flexibles, tomando en cuenta no solo el emparejamiento exacto de entidades, sino también sus ancestros y descendientes dentro del grafo de conocimiento.

### 7.2.5. Soporte para múltiples formatos de datos

Actualmente, el sistema solo admite tablas en formato CSV y archivos de metadatos en formato JSON. Sería conveniente abstraerse del formato de archivo en sí, de modo que se puedan implementar diferentes estrategias de preprocesamiento para adaptarse a cada tipo de fuente de datos.

### 7.2.6. Entorno con mayores capacidades

El entorno de trabajo impuso varios límites en el desarrollo del proyecto. Por ejemplo, como se mencionó anteriormente, no fue posible contar con una copia local de la base de conocimiento, lo que restringió algunas optimizaciones. Además, debido a la alta demanda de recursos, no se pudieron probar modelos LLM más avanzados.

### 7.2.7. Interfaz visual para evaluación humana

Sería de utilidad la implementación de una interfaz visual que permita realizar una evaluación manual de los resultados obtenidos, tanto en lo referente a los metadatos generados como a las relaciones detectadas. La idea es incentivar la retroalimentación humana (RHLF) en el sistema, lo que contribuirá a su validación y mejora contínua.

### 7.2.8. Implementación del mantenimiento del sistema

Aunque en la sección 5.6 se especificaron las tareas necesarias para que el sistema soporte la ingesta, actualización y borrado de conjuntos de datos, esto no fue implementado por motivos de tiempo. Estas funcionalidades son indispensables si se quiere utilizar la herramienta para aplicaciones reales.

### 7.2.9. Relaciones semánticas entre columnas de una misma tabla

La versión original de TableMiner+ descrita en 3.3 incluye la anotación de relaciones entre columnas de una misma tabla con propiedades de una base de conocimiento. Sin embargo, en la implementación de la solución de base esta funcionalidad no estaba incluida y si bien no fue incorporada por motivos de alcance del proyecto, resultaría de gran utilidad para desambiguar aún más los recursos.

# Bibliografía


[1] What is data governance? Último acceso: 16 de marzo de 2025. URL: https://cloud.google.com/learn/what-is-data-governance.

[2] AGESIC. Catálogo nacional de datos abiertos de uruguay. Último acceso: 28 de diciembre de 2024. URL: https://catalogodatos.gub.uy/.

[3] AGESIC. Catálogo Nacional de Datos Abiertos de Uruguay. Último acceso: 5 de enero de 2025. URL: https://catalogodatos.gub.uy/about.

[4] AGESIC. Catálogo Nacional de Datos Abiertos de Uruguay. Último acceso: 5 de enero de 2025. URL: https://catalogodatos.gub.uy/agesic/portal/panel.

[5] Blazegraph. Welcome to blazegraph. Último acceso: 10 de enero de 2025. URL: https://blazegraph.com/.

[6] A. Bogatu, A.A.A. Fernandes, N.W. Paton, and N. Konstantinou. Dataset discovery in data lakes. In *Proceedings - International Conference on Data Engineering*, volume 2020-April, pages 709–720, 2020. doi:10.1109/ICDE48307.2020.00067.

[7] CKAN. API guide. Último acceso: 04 de marzo de 2025. URL: https://docs.ckan.org/en/2.9/api/#api-examples.

[8] CKAN. Ckan: The open source data portal software. Último acceso: 28 de diciembre de 2024. URL: https://ckan.org/.

[9] Mamata Das, Selvakumar K., and P. J. A. Alphonse. A comparative study on tf-idf feature weighting method and its analysis using unstructured dataset, 2023. URL: https://arxiv.org/abs/2308.04037, arXiv:2308.04037.

[10] G. Fan, J. Wang, Y. Li, D. Zhang, and R.J. Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *Proceedings of the VLDB Endowment*, 16:1726–1739, 2023. doi:10.14778/3587136.3587146.

[11] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: a data discovery system. In *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*, pages 1001–1012, 2018. doi:10.1109/ICDE.2018.00094.

[12] Wikimedia Foundation. wikimedia foundation. Último acceso: 23 de enero de 2025. URL: https://wikimediafoundation.org/es/.

[13] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL: https://arxiv.org/abs/2312.10997, arXiv:2312.10997.

[14] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35:12571–12590, 12 2023. doi:10.1109/TKDE.2023.3270101.

[15] Efthymiou V. Chen J. Hassanzadeh, O. Semtab 2023: Semantic web challenge on tabular data to knowledge graph matching data sets - "wikidata tablesr1, 2023. doi:10.5281/zenodo.8393535.

[16] Oktie Hassanzadeh, Nora Abdelmageed, Vasilis Efthymiou, Jiaoyan Chen, Vincenzo Cutrona, Madelon Hulsebos, Ernesto Jiménez-Ruiz, Aamod Khatiwada, Keti Korini, Benno Kruit, Juan Sequeda, and Kavitha Srinivas. Results of semtab 2023. 2023. URL: https://ceur-ws.org/Vol-3557/paper0.pdf.

[17] Oktie Hassanzadeh, Vasilis Efthymiou, and Jinru Chen. SemTab 2023: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets - "Wikidata TablesR1 (2023 (Wikidata Tables, R1)), 2023. doi:10.5281/zenodo.8393535.

[18] HuggingFace. all-mpnet-base-v2. Último acceso: 04 de marzo de 2025. URL: https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

[19] Norman W. Paton, Jiaoyan Chen, and Zhenyu Wu. Dataset discovery and exploration: A survey. *ACM Computing Surveys*, 56, 11 2023. doi:10.1145/3626521.

[20] Norman W. Paton and Zhenyu Wu. Dataset discovery and exploration: State-of-the-art, challenges and opportunities. In *Advances in Database Technology - EDBT*, volume 27, pages 854–857. OpenProceedings.org, 3 2024. doi:10.48786/edbt.2024.87.

[21] Norman W. Paton and Zhenyu Wu. Dataset discovery and exploration: State-of-the-art, challenges and opportunities, 2024. URL: https://openproceedings.org/2024/conf/edbt/tutorial-3.pdf.

[22] Mark Pilgrim and Ian Cordasco. chardet: The universal character encoding detector. https://github.com/chardet/chardet. Último acceso: 5 de enero de 2025.

[23] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020. URL: https://arxiv.org/abs/2010.16061, arXiv:2010.16061.

[24] Pegdwendé Sawadogo and Jérôme Darmont. On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, 56:97–120, 2 2021. doi:10.1007/s10844-020-00608-7.

[25] Welcome to Wikidata. Wikidata. Último acceso: 14 de enero de 2025. URL: https://www.wikidata.org/wiki/Wikidata:Main_Page.

[26] Oxford University. Semtab: Semantic web challenge on tabular data to knowledge graph matching. Accedido: 2025-01-12. URL: https://www.cs.ox.ac.uk/isg/challenges/sem-tab/.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] W3C. RDF 1.0 XML Syntax Specification. Último acceso: 16 de marzo de 2025. URL: https://www.w3.org/TR/rdf10-xml/.

[29] Wikidata. Wikidata: Introduction. Último acceso: 23 de enero de 2025. URL: https://www.wikidata.org/wiki/Wikidata:Introduction.

[30] Z. Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8:921–957, 2017. doi:10.3233/SW-160242.

[31] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2024. URL: https://arxiv.org/abs/2303.18223, arXiv:2303.18223.

# Anexo A

# Definiciones

## A.1. Hashing Sensible a la Localidad (LSH)

LSH es una técnica para realizar búsquedas de vecinos más cercanos en espacios de alta dimensionalidad. Esta técnica requiere funciones de *hash* cuya probabilidad de colisión sea alta para entradas similares y más baja para entradas diferentes.
En un índice LSH, el grado de similitud entre dos elementos se mide por el número de contenedores (*buckets*), es decir, entradas del índice, que contienen ambos elementos. El tipo de similitud alcanzado depende de la función de *hash* utilizada [6].

### A.1.1. MinHash

MinHash es una función de *hash* que permite estimar la similitud de Jaccard entre conjuntos sin necesidad de computar explícitamente sus intersecciones y uniones. Se basa en la generación de firmas compactas para cada conjunto mediante funciones de *hash* independientes, donde se conserva el menor valor obtenido en cada función. La similitud entre dos conjuntos se aproxima comparando la fracción de valores mínimos compartidos en sus respectivas firmas [6].

#### Similitud de Jaccard

La similitud de Jaccard es una medida matemática ampliamente utilizada en el análisis de conjuntos para evaluar el grado de similitud y diversidad entre ellos. Su cálculo se basa en la relación entre el tamaño de la intersección y el tamaño de la unión de los conjuntos comparados. Formalmente, se define como:

$$\mathcal{J}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- **A** y **B:** conjuntos arbitrarios.
- **A** $\cap$ **B:** elementos comunes a ambos conjuntos.
- **A** $\cup$ **B:** elementos presentes en al menos uno de los conjuntos.

El valor de la similitud de Jaccard se encuentra en el rango de 0 a 1. Un resultado de 1 indica que los conjuntos son completamente idénticos, mientras que un valor de 0 señala que no comparten elementos.

### A.1.2. Proyecciones aleatorias

Las proyecciones aleatorias son una técnica de LSH utilizada para reducir la dimensionalidad de los datos manteniendo relaciones de similitud coseno. Consisten en proyectar los vectores originales sobre hiperplanos aleatorios, utilizando el producto punto con vectores de proyección generados aleatoriamente. En cada proyección, se almacena únicamente el signo del resultado, formando una firma *hash* binaria [6].

#### Similitud coseno

La similitud de coseno es una medida utilizada para analizar la relación entre dos vectores en un espacio vectorial multidimensional. Su cálculo se basa en el coseno del ángulo que forman los vectores, ignorando su magnitud. Formalmente, se define como:

$$S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **A · B:** Representa el producto escalar de los vectores A y B.
- **||A|| y ||B||:** Son las normas euclidianas de los vectores A y B.
- **cos($\theta$):** Indica la orientación relativa de los vectores.

El valor de la similitud de coseno varía entre -1 y 1. Un valor de 1 indica vectores perfectamente alineados, 0 señala vectores ortogonales y -1 refleja vectores opuestos.

## A.2. TF-IDF

TF-IDF es una métrica ampliamente utilizada en el procesamiento de lenguaje natural para evaluar la importancia de un término dentro de un documento y en relación con un conjunto de documentos [9]. Combina dos componentes fundamentales:

1. **Frecuencia de Aparición de Términos (TF):** La frecuencia de términos mide la proporción de veces que un término aparece en un documento, en comparación con el total de términos en el mismo. Esto permite identificar la representatividad del término en relación con el documento individual. Se calcula como:

   $$TF(t, d) = \frac{f_{t,d}}{\sum_k f_{k,d}}$$

   Donde:

   - $f_{t,d}$: Frecuencia del término $t$ en el documento $d$.

- $\sum_k f_{k,d}$: Total de términos en el documento $d$.

2. **Frecuencia Inversa de Documentos (IDF):** La IDF mide la rareza de un término en el *corpus* completo. Términos que aparecen en muchos documentos tienen un menor peso, mientras que términos más únicos reciben mayor importancia. Se calcula como:

$$IDF(t, D) = \log\left(\frac{N}{1 + |d \in D : t \in d|}\right)$$

Donde:

- $N$: Total de documentos en el corpus $D$.
- $|\{d \in D : t \in d\}|$: Cantidad de documentos que contienen el término $t$.

3. **Peso TF-IDF:** El peso TF-IDF combina los valores de TF e IDF para determinar la importancia de un término dentro de un documento, teniendo en cuenta el corpus completo. Se calcula como:

$$TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Este cálculo es especialmente útil para identificar términos que son significativos dentro de un documento, pero que no están ampliamente distribuidos en el corpus.

## A.3. Base de conocimiento

Según [30], una base de conocimiento define un conjunto de conceptos, entidades, literales que representan valores de datos concretos, y relaciones semánticas que definen posibles asociaciones entre entidades (y, por lo tanto, también entre los conceptos a los que pertenecen) o entre una entidad y un literal. En este último caso, la relación suele llamarse una propiedad de la entidad (y, por lo tanto, una propiedad de su concepto) y el literal se denomina el valor de la propiedad.
En su forma genérica, una base de conocimiento es un conjunto de datos enlazados que contiene un conjunto de triplas, declaraciones o hechos, cada uno compuesto por un sujeto, un predicado y un objeto.

### A.3.1. Wikidata

Según [29], Wikidata es una base de conocimiento libre, colaborativa y multilingüe que recopila datos estructurados para brindar apoyo a los proyectos de Wikimedia [12] y a cualquier persona en el mundo. Al ser libre y abierta, puede ser leída y editada tanto por personas como por máquinas [25].

#### Estructura y Organización de Datos

El repositorio de Wikidata consiste principalmente en "elementos". Cada elemento tiene una etiqueta, una descripción y es identificado de manera única con una "Q" seguida de un número (por ejemplo, Q77 para Uruguay).

Además, los elementos pueden estar relacionados entre sí a través de propiedades. Las declaraciones en Wikidata describen características específicas de un elemento, estableciendo una relación

entre una propiedad y un valor. Las propiedades se identifican con una “P” seguida de un número.

Toda esta información es accesible en cualquier idioma, incluso si los datos originales fueron ingresados en otro. Al consultar estos valores, siempre se obtiene la versión más actualizada de la información.

# Anexo B

# Ejemplos de datos del Catálogo

En este anexo se presentan ejemplos de archivos de datos y metadatos del Catálogo de Datos Abiertos de Uruguay.

## B.1. Archivo de datos

Ejemplo de estructura del archivo de datos CSV que se encuentra en el catálogo de datos.

| OID | Tipo Organismo | Inciso | Nombre Inciso | Nombre UE | Tipo de Información | Cantidad |
|---|---|---|---|---|---|---|
| 2.16.858.0.0.2.6 | Servicio Descentralizado | 67 | Adm. Nacional de Correos | Adm. Nacional de Correos | Personal de Ventanilla | 275 |
| 2.16.858.0.0.2.6 | Servicio Descentralizado | 67 | Adm. Nacional de Correos | Adm. Nacional de Correos | Carteros | 709 |
| 2.16.858.0.0.2.6 | Servicio Descentralizado | 67 | Adm. Nacional de Correos | Adm. Nacional de Correos | Locales Propios | 121 |
| 2.16.858.0.0.2.6 | Servicio Descentralizado | 67 | Adm. Nacional de Correos | Adm. Nacional de Correos | Agentes Oficiales | 126 |
| 2.16.858.0.0.2.6 | Servicio Descentralizado | 67 | Adm. Nacional de Correos | Adm. Nacional de Correos | Centros de Atención Ciudadana | 2 |

Figura B.1: Extracto del dataset 'Sobre Correo Uruguayo'.

## B.2. Archivo de metadatos

Ejemplo de estructura del archivo de metadatos JSON que se encuentra en el catálogo de datos.

```
{
    "atributos": [
        {
            "descripcion": "Identificador Único del Organismo",
            "tipoDeDato": "String",
            "nombreDeAtributo": "OID"
        },
        {
            "descripcion": "Tipo de organización pública",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Tipo Organismo"
        },
        {
            "descripcion": "Número de inciso",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Inciso"
        },
        {
            "descripcion": "Unidad Ejecutora",
            "tipoDeDato": "String",
            "nombreDeAtributo": "UE"
        },
        {
            "descripcion": "Nombre del Inciso",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Nombre Inciso"
        },
        {
            "descripcion": "Nombre de la Unidad Ejecutora",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Nombre UE"
        },
        {
            "descripcion": "Nombre del Organismo",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Nombre Organismo"
        },
        {
            "descripcion": "Locales comerciales y flota vehicular",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Tipo de informacion"
        },
        {
            "descripcion": "Cantidad de personal y locales",
            "tipoDeDato": "Numeric",
            "nombreDeAtributo": "Cantidad"
        },
        {
            "descripcion": "Descripción de los datos",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Descripcion"
        },
        {
            "descripcion": "Enlace a la información",
            "tipoDeDato": "String",
            "nombreDeAtributo": "Enlace"
        }
    ],
    "titulo": "Metadatos de información sobre el organismo",
    "descripcion": "Metadatos de información sobre el organismo"
}
```


Figura B.2: Metadatos JSON.

# Anexo C

# Prompts para el LLM

En este anexo se presentan las *prompts* al LLM que se utilizaron en las diversas tareas de generación de texto.

## C.1. Generación de descripción

*Prompt* utilizada para generar una descripción de un conjunto de datos, a través de información de la tabla y el recurso.

```
Eres un asistente que ayuda en la desambiguación de tablas.
Toda la información pertenece al catálogo de datos abierto de Uruguay.

### Instrucciones
- Solo se debe generar como output descripciones detalladas y específicas.
- No uses frases genéricas como "No hay datos relevantes".
  Omitir en la respuesta todo lo que no sea una descripción.
- Solo responder con la descripción, ser lo más objetivo posible.

### Ejemplo 1
- Nombre tabla: <nombre_tabla_1>
- Nombre recurso: <nombre_recurso_1>
- Tabla: <20_filas_de_la_tabla_1>

- Descripción de salida: <descripción_de_salida_1>

... más few shots ...

### Ahora genera una descripción para la siguiente tabla:
- Nombre tabla: <nombre_tabla>
- Nombre recurso: <nombre_recurso>
- Tabla: <20_filas_de_la_tabla>

- Descripción de salida:
```

Figura C.1: *Prompt* para generar descripción

## C.2. Generación de descripción usando archivo de metadatos

*Prompt* utilizada para generar una descripción de un conjunto de datos, a través de información de la tabla, el recurso, y de un archivo con metadatos.

```
Eres un asistente que ayuda en la desambiguación de tablas.
Toda la información pertenece al catálogo de datos abierto de Uruguay.

### Instrucciones
- Solo se debe generar como output descripciones detalladas y específicas.
- No uses frases genéricas como "No hay datos relevantes".
  Omitir en la respuesta todo lo que no sea una descripción.
- Solo responder con la descripción, ser lo más objetivo posible.

### Ejemplo 1
- Nombre tabla: <nombre_tabla_1>
- Nombre recurso: <nombre_recurso_1>
- Contexto: <descripcion_general_1>
- Tabla: <20_filas_de_la_tabla_1>
- Archivo de metadatos: <archivo_de_metadatos_1>

- Descripción de salida: <descripción_de_salida_1>

... más few shots ...

### Ahora genera una descripción para la siguiente tabla:
- Nombre tabla: <nombre_tabla>
- Nombre recurso: <nombre_recurso>
- Contexto: <descripcion_general>
- Tabla: <20_filas_de_la_tabla>
- Archivo de metadatos: <archivo_de_metadatos>

- Descripción de salida:
```

Figura C.2: Prompt para generar descripción con archivo de metadatos

## C.3. Generación de archivo de metadatos

Utilizada para generar una descripción de una columna en particular. Esta *prompt* es la encargada de generar la información necesaria para poder crear el archivo de metadata descriptiva de una determinada tabla. Usa información de la tabla, del recurso, y la descripción general.

```
Eres un asistente que ayuda en la desambiguación de tablas.
Toda la información pertenece al catálogo de datos abierto de Uruguay.

### Instrucciones
- Solo se debe generar como output descripciones detalladas y específicas.
- No uses frases genéricas como "No hay datos relevantes".
  Omitir en la respuesta todo lo que no sea una descripción.
- Solo responder con la descripción, ser lo más objetivo posible.

### Ejemplo 1
- Nombre tabla: <nombre_tabla_1>
- Nombre recurso: <nombre_recurso_1>
- Contexto: <descripcion_general_1>
- Tabla: <20_filas_de_la_tabla_1>
- Columna de interés: <nombre_columna_1>

- Descripción de salida de la columna de interés: <descripción_de_salida_columna_j>

... más few shots ...

### Ahora genera una descripción para la siguiente columna:
- Nombre tabla: <nombre_tabla>
- Nombre recurso: <nombre_recurso>
- Contexto: <descripcion_general>
- Tabla: <20_filas_de_la_tabla>
- Columna de interés: <nombre_columna>

- Descripción de salida de la columna de interés:
```

Figura C.3: Prompt para generar archivo de metadatos con descripción

## C.4. Generación de concepto candidato para columna

Esta *prompt* es utilizada en el contexto de TableMiner+ como mecanismo de *fallback*, para poder generar un concepto candidato para una columna en caso de que no se haya podido encontrar.

### C.4.1. Ejemplo 1

En esta *prompt* se utiliza toda la información disponible del paquete.

```
Eres un asistente que ayuda en la desambiguación de tablas.
Toda la información pertenece al catálogo de datos abierto de Uruguay.

### Instrucciones
- Solo se debe generar como output descripciones detalladas y específicas.
- No uses frases genéricas como "No hay datos relevantes".
  Omitir en la respuesta todo lo que no sea una descripción.
- Solo responder con la descripción, ser lo más objetivo posible.

### Ejemplo 1
- Nombre columna: <nombre_columna_1>
- Ejemplos de valores: <valores_de_la_columna>
- Nombre tabla: <nombre_tabla_1>
- Nombre recurso: <nombre_recurso_1>
- Contexto: <descripcion_general_1>
- Archivo de metadatos: <archivo_de_metadatos_1>
- Algunas filas de la tabla: <5_filas_aleatorias>

- Concepto sugerido: <concepto_sugerido_1>

... más few shots ...

### Ahora genera un concepto para la siguiente columna de la tabla:
- Nombre columna: <nombre_columna>
- Ejemplos de valores: <valores_de_la_columna>
- Nombre tabla: <nombre_tabla>
- Nombre recurso: <nombre_recurso>
- Contexto: <descripcion_general>
- Archivo de metadatos: <archivo_de_metadatos>
- Algunas filas de la tabla: <5_filas_aleatorias>

- Concepto sugerido:
```

Figura C.4: Prompt para generar descripción con archivo de metadatos

### C.4.2. Ejemplo 2

En esta *prompt* se omite el contexto y se utiliza principalmente información de la columna que debe anotar.

```
Eres un asistente experto en datos especializado en categorizar
columnas de datos utilizando conceptos de Wikidata.
Tu tarea es analizar **todos los valores de la columna** y
sugerir **un único concepto de Wikidata** que los represente
de la mejor manera posible.
- Considera todos los valores proporcionados, no solo un
subconjunto de ellos.
- Devuelve **únicamente** la etiqueta del concepto de Wikidata
(por ejemplo, "medical treatment").
- **No** expliques el razonamiento.
- La respuesta debe ser un solo concepto de Wikidata.

### Ejemplos

#### Ejemplo 1:
Nombre de la Tabla: <nombre_tabla_1>
Nombre Columna: <nombre_columna>
Valores:
<listado_valores_columna>

### Concepto sugerido:
<concepto_sugerido_1>

... más few shots ...

### Ahora, analiza la siguiente columna y tabla para generar un concepto de Wikidata:
Nombre de la Tabla: <nombre_tabla>
Nombre Columna: <nombre_columna>
Valores:
<listado_valores_columna>

### Concepto sugerido:
```

Figura C.5: Prompt para generar descripción con archivo de metadatos

## C.5. Respuesta a consulta en lenguaje natural

*Prompt* utilizada para la tarea de RAG, donde se especifica como es deseable que el modelo se comporte para devolver la respuesta a la consulta del usuario.

```
### Instrucciones:
    - Ser amigable, responder la pregunta del usuario:"{query}".
      La información debería poder encontrarse en los recursos a continuación.
    - Adherirse a la información dada y no inventar.
      Se puede inferir conocimiento solo si es obvio.
    - La respuesta es para un usuario buscando recursos de su interés.
    - Generar la respuesta que se mostrará al usuario en lenguaje natural
      a partir de "### Respuesta", ser conciso.
    - La idea es que el usuario solo reciba una respuesta, y se le sugieran
      los recursos listados a continuación.

### Información para usar en la respuesta:
#### Recurso 1:
    - Título: <titulo_dataset>
    - Organización: <organizacion_publicadora>
    - Detalles: <descripcion_dataset>
    - URL del CSV con los datos: <url_del_recurso_1>

...más recursos...

#### Recurso extra:
    - Título: <titulo_dataset>
    - Organización: <organizacion_publicadora>
    - Detalles: <descripcion_dataset>
    - URL del CSV con los datos: <url_del_recurso>

Ahora genera la respuesta para la query del usuario, usando
la informacion dada arriba: {query}

### Respuesta
```

Figura C.6: Prompt para RAG

# Anexo D

# Consultas SPARQL a la base de conocimiento

En este anexo se presentan las consultas SPARQL realizadas a la base de conocimiento para buscar entidades y conceptos durante la anotación en TableMiner+. Estas consultas son las utilizadas en la implementacion de TableMiner+ del trabajo [21] para Wikidata.

## D.1. Buscar entidad dado el contenido de la celda

Esta consulta busca entidades en Wikidata cuyas etiquetas coincidan parcialmente con el contenido de una celda dada.

```
SELECT ?item ?itemLabel WHERE {
  ?item rdfs:label ?itemLabel .
  FILTER(LANG(?itemLabel) = "es")
  FILTER(CONTAINS(LCASE(?itemLabel), LCASE("{cell_content}")))
}
LIMIT {limit}
```

Figura D.1: Consulta para obtener entidades dado el contenido de una celda

## D.2. Buscar triplas asociadas a una entidad

Esta consulta permite recuperar las propiedades y valores asociados a una entidad específica en Wikidata. Es útil para extraer información estructurada sobre una entidad y sus relaciones en la base de conocimiento.

```
SELECT ?property ?propertyLabel ?value ?valueLabel WHERE {
  BIND(wd:{entity_id} AS ?entity)
  ?entity ?p ?statement .
  ?statement ?ps ?value .

  ?property wikibase:claim ?p.
  ?property wikibase:statementProperty ?ps.

  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
}
```

Figura D.2: Consulta para obtener triplas dada una entidad

## D.3. Buscar conceptos asociados a una entidad

Esta consulta permite recuperar conceptos (tipos semánticos) asociados a una entidad específica en Wikidata.

```
SELECT ?concept ?conceptLabel WHERE {
  wd:Q42 wdt:P31/wdt:P279? ?concept .
  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
}
```

Figura D.3: Consulta para obtener conceptos dada una entidad

## D.4. Buscar descripción corta de una entidad

Esta consulta permite recuperar la descripción breve de una entidad específica en Wikidata, utilizando la propiedad schema:description.

```
SELECT ?entityDescription WHERE {
    wd:Q42 schema:description ?entityDescription.
    FILTER(LANG(?entityDescription) = "en")
}
```

Figura D.4: Consulta para obtener la descripción de una entidad

# Anexo E

# Guía para la ejecución de la herramienta implementada

En esta sección se provee una guía para ejecutar la herramienta implementada. Por un lado, se explicitan los requerimientos del sistema necesarios, y por otro, se muestran y se explica el contenido de los directorios principales del código base. Al final, se da una guía para la ejecución de la herramienta paso a paso. El código base de la herramienta se puede ver en el repositorio de Github de referencia[1].

Para poder ejecutar la herramienta localmente se debe contar con:

- Python 3.10.11 o posteriores.
- Manejador de dependencias pip 25.0.
- Entorno de ejecución de Jupyter Notebook. En particular, se recomienda utilizar el editor de texto Visual Studio Code[2] en conjunto con la extensión de Jupyter[3].
- Unidad de procesamiento gráfica que soporte CUDA[4] (versión 12.6). Este requerimiento es necesario en caso de que se quiera utilizar el LLM para generación de metadatos, RAG o como *fallback* de TableMiner+.

En la figura E.1 se observa como está estructurado el árbol de directorios del proyecto. En particular, el archivo principal es el *notebook* `DataDiscovery.ipynb`, donde se llevan a cabo todas las etapas del procesamiento de los *datasets*, hasta la salida final. Por otro lado, se tienen carpetas dedicadas a cada uno de los distintos módulos:

- `Aurum`: Este directorio contiene la implementación de Aurum.
- `D3L`: Este directorio contiene la implementación de D3L.

[1] https://github.com/Nicolasvb23/proyecto-de-grado
[2] https://code.visualstudio.com/
[3] https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter
[4] https://developer.nvidia.com/cuda-toolkit

- `DatasetsUtils`: Este directorio contiene los *scripts* de descarga a través de *tags* utilizando la API de CKAN, junto con todas las tareas de preprocesamiento y extracción de medidas de la calidad de los datos.
- `MetadataLLM`: En este directorio se encuentran todos los *prompts* que se realizan al LLM, tanto los de la generación de metadatos como el del mecanismo de *fallback* de TableMiner+.
- `TableMiner+`: En esta carpeta se encuentra la implementación de TableMiner+.
- `ApiCatalogoCKAN.ipynb`: En este *notebook* se provee un entorno de pruebas para utilizar la API provista por CKAN. Se tienen diferentes celdas las cuales realizan peticiones a los diferentes *endpoints* disponibles, imprimiendo la respuesta.

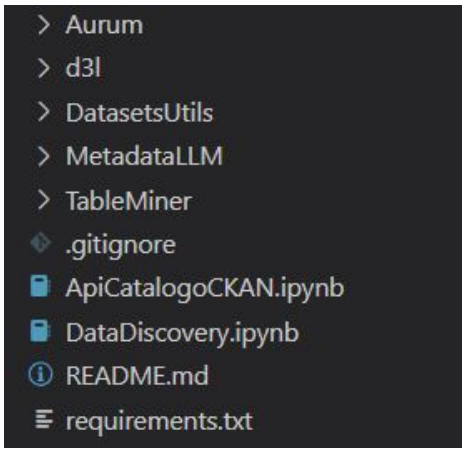


Figura E.1: Árbol de directorios del proyecto.

Para poder ejecutar la herramienta, como primer paso, se deben instalar todas las dependencias del proyecto. Estas se encuentran especificadas, junto con su versión, en el archivo `requirements.txt`:

```
$ pip install -r requirements.txt
```

Figura E.2: Comando de instalación de las dependencias necesarias en la herramienta.

Una vez instaladas las dependencias, se puede comenzar a ejecutar el *notebook* principal `DatasetDiscovery.ipynb`. Los módulos se ejecutan en este orden:

1. **Descarga, procesamiento y extracción de medidas de *datasets* del catálogo**: Se puede especificar el conjunto de palabras de interés para la descarga a través de la modificación de la variable `interest_words`. Una vez terminada esta etapa, se habrá creado un directorio con el nombre de `PipelineDatasets` el cual contiene tres directorios. Uno para los *datasets* recién descargados sin aplicar ningún procesamiento adicional (`DownloadedDatasets`), otro para los *datasets* luego del preprocesamiento (`DatasetsCollection`), y finalmente los *datasets* luego

de la selección automática (`SelectedDatasets`). Por otro lado, se crea un directorio `Datasets` que contiene solo los recursos CSV de los conjuntos de datos que fueron seleccionados.

2. **Búsqueda de conjuntos de datos (D3L)**: Una vez generados los índices, estos se guardarán en un archivo *pickle*[5] dentro del directorio `Results`. Si se cierra la sesión del *notebook* y se quiere volver a ejecutar las celdas, en caso de que existan índices LSH guardados en este directorio se cargarán y se omitirá la generación de nuevos índices. Si se quiere comenzar desde cero, es recomendable borrar este directorio.

3. **Navegación de datos (Aurum)**: Como resultado, dará una visualización de un grafo, mostrando las relaciones entre *datasets* que fueron identificadas a través de D3L.

4. **Carga del LLM**: En esta etapa se realiza la carga del LLM. Se debe omitir en caso de que no se cumpla con los requisitos para cargar el modelo de lenguaje localmente. De lo contrario, para poder acceder a los modelos gratuitos que provee HuggingFace se debe contar con un token de acceso el cual se debe generar utilizando una cuenta en el sitio web[6].

5. **Anotación de datos (TableMiner+)**: Como salida se genera el archivo `annotationDict.json` en la carpeta `Results`. Este archivo contiene todas las anotaciones semánticas obtenidas de la base de conocimiento, y los tipos de datos de las columnas.

6. **Generación de metadatos con el LLM**: Genera dos nuevos directorios dentro de `PipelineDatasets`:

   - `EnrichedDatasets` que contiene los metadatos generados.
   - `FinalDatasets` que contiene los conjuntos de datos originales junto con los metadatos generados ya incorporados.

7. **Mecanismo RAG**: Requiere del LLM para funcionar. En caso de que se disponga de los recursos para cargar el LLM, solo hace falta cambiar la *query* para hacer la consulta deseada al mecanismo de *Retrieval Augmented Generation*.

[5] https://docs.python.org/es/3.13/library/pickle.html
[6] https://huggingface.co/settings/tokens