Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento del Lenguaje Natural

“Trabajo Práctico Final”

Autora: Valentina Balverdi - (B-6588/9)

Profesores:
- Juan Pablo Manson
- Alan Geray
- Constantino Ferrucci

Fecha de entrega: 7 de diciembre

2025

Introducción

El presente trabajo práctico tiene como objetivo desarrollar un asistente virtual capaz de responder preguntas relacionadas con productos de una empresa de electrodomésticos utilizando técnicas de Procesamiento del Lenguaje Natural (NLP). Para ello, se integran distintos tipos de datos y modelos con el fin de implementar un sistema de Recuperación con Generación Aumentada (RAG) y, posteriormente, un agente autónomo basado en el paradigma ReAct.

El trabajo se divide en dos ejercicios principales.
En el Ejercicio 1, se construye un sistema RAG que combina tres fuentes independientes de información:

- una base vectorial con documentos textuales,
- una base tabular con datos estructurados, y
- una base de grafos que modela relaciones entre productos.

Además, se implementa un clasificador de intención para determinar la fuente adecuada para cada consulta y un pipeline de recuperación que combina búsqueda semántica e instrucciones generadas por un modelo de lenguaje.

En el Ejercicio 2, el sistema se amplía mediante la creación de un agente autónomo capaz de seleccionar herramientas especializadas y ejecutar consultas más complejas. Dicho agente utiliza LangChain y el enfoque ReAct para razonar, elegir acciones y combinar información de distintas fuentes de manera dinámica.

Este informe describe los objetivos, la metodología, el desarrollo y los resultados obtenidos en el Ejercicio 1, junto con las conclusiones generales del trabajo.

Ejercicio 1 – RAG: Sistema de Recuperación con Generación Aumentada

Objetivo del Ejercicio 1

El objetivo del Ejercicio 1 es desarrollar un sistema RAG capaz de responder consultas sobre productos de una empresa de electrodomésticos utilizando tres fuentes de datos complementarias:

- Vectorial: documentos textuales procesados mediante embeddings.
- Tabular: información estructurada en DataFrames.
- Grafo: relaciones entre productos representadas mediante nodos y aristas.

El sistema debe seleccionar de manera adecuada la fuente correspondiente a cada consulta y devolver una respuesta basada en la información recuperada. Para ello, se requiere:

1. construir las tres bases de datos,

2. implementar un clasificador de intención que determine qué fuente utilizar,
3. diseñar un pipeline de recuperación que combine búsqueda semántica y consultas dinámicas, y
4. establecer una interfaz que permita integrar la recuperación con un modelo de lenguaje generativo.

Metodología del Ejercicio 1

La metodología del Ejercicio 1 se centra en la construcción de un sistema RAG que utilice múltiples fuentes de información y permita responder consultas de lenguaje natural de forma precisa. Para ello, el desarrollo se realizó íntegramente en Google Colab y empleó herramientas de procesamiento de texto, embeddings, recuperación semántica, consultas tabulares y modelado de grafos.

Entorno y herramientas

El trabajo se implementó en un cuaderno de Google Colab, utilizando las siguientes librerías principales:

- pandas para la manipulación de datos tabulares.
- ChromaDB para la base vectorial.
- scikit-learn para la construcción del clasificador de intención.
- LangChain para la conversión de consultas a código y para el manejo de LLMs.
- Modelos de lenguaje alojados en la nube, utilizados para producir filtros dinámicos, generar consultas y sintetizar respuestas.

Datos utilizados

Se utilizaron los conjuntos de datos provistos en el enunciado del trabajo práctico, entre ellos:

- Documentos textuales (manuales, reseñas, FAQs y tickets).
- Tablas estructuradas con información de productos, ventas y vendedores.
- Un DataFrame creado con relaciones entre productos utilizado para el grafo.

Se trabajó únicamente con aquellos necesarios para las fuentes vectorial, tabular y de grafos.

Construcción de las tres fuentes de datos

a) Fuente vectorial

Los documentos textuales fueron procesados mediante:

- un modelo de embeddings en español,
- un text splitter para dividir los textos en fragmentos, y
- una base vectorial en ChromaDB para almacenar los embeddings y permitir búsquedas por similitud.

Se desarrolló una función de consulta que recibe una pregunta del usuario y devuelve los fragmentos más relevantes según distancia semántica.

b) Fuente tabular

Los datos estructurados fueron cargados en DataFrames. Para permitir consultas dinámicas sin introducir las tablas completas en el contexto del modelo, se extrajeron metadatos relevantes tales como:

- valores únicos,
- rangos numéricos,
- categorías presentes en cada tabla.

Con esta información se construyó un prompt especializado, utilizado por un modelo de lenguaje para generar filtros en pandas. Estos filtros se ejecutan sobre los Data Frames y producen la respuesta correspondiente.

c) Fuente de grafos

A partir de un DataFrame con relaciones entre productos, se preparó la estructura necesaria para construir un grafo consultable mediante Cypher. La metodología incluye:

- definición de nodos (productos),
- definición de relaciones (compatibilidad u otros vínculos),
- inserción en una base que soporte consultas Cypher,
- uso de un modelo de lenguaje para convertir consultas en lenguaje natural en instrucciones Cypher ejecutables.

Desarrollo del Ejercicio 1

En esta sección se describen los pasos de implementación del sistema RAG correspondiente al Ejercicio 1, detallando la construcción de las tres fuentes de información, el clasificador de intención y el pipeline de recuperación.

Implementación de la base vectorial

Los documentos textuales (manuales, reseñas de usuarios, FAQs y tickets de soporte) fueron procesados previamente para unificarlos en un formato consistente. Cada documento fue dividido en fragmentos mediante el RecursiveCharacterTextSplitter, configurado con un tamaño de 512 caracteres y un solapamiento de 64. Esta elección responde a la naturaleza heterogénea del corpus: las reseñas, FAQs y tickets son textos breves, mientras que los manuales contienen secciones extensas.

El tamaño seleccionado permite preservar los textos cortos dentro de un único fragmento y, al mismo tiempo, segmentar los documentos largos en unidades manejables sin pérdida de coherencia. El solapamiento garantiza que las ideas ubicadas cerca de los límites entre fragmentos no se vean interrumpidas, favoreciendo la continuidad semántica en el proceso de recuperación. El uso del splitter recursivo permite además realizar cortes en puntos

naturales (doble salto de línea, punto, espacio), generando fragmentos más legibles y adecuados para el modelo de recuperación.

Posteriormente, se empleó el modelo de embeddings paraphrase-multilingual-mpnet-base-v2 para transformar cada fragmento en una representación vectorial. Su selección se justifica por tres razones: ofrece un desempeño robusto en español, produce embeddings semánticamente expresivos incluso ante consultas parafraseadas y presenta un costo computacional compatible con el entorno de ejecución en Google Colab.

Los vectores obtenidos se almacenaron en una colección de ChromaDB, permitiendo realizar búsquedas por similitud mediante distancia coseno. Para acceder a esta fuente se desarrolló una función de consulta que recibe una pregunta en lenguaje natural y un valor *k*, y devuelve los *k* fragmentos más relevantes según su proximidad semántica. Esta función constituye la interfaz principal de la base vectorial dentro del sistema RAG.

Implementación de la fuente tabular

Los datos estructurados se cargaron en distintos DataFrames de pandas. Para evitar pasar tablas enteras al contexto del modelo de lenguaje, se extrajeron metadatos relevantes tales como listas de valores únicos para variables categóricas, rangos y valores mínimos y máximos en variables numéricas, categorías y descripciones clave de productos.

Con esta información se diseñó un prompt especializado, dirigido a un LLM que actúa como un “experto en análisis de datos con pandas”. A partir de una pregunta del usuario, este modelo genera los filtros adecuados para seleccionar los datos, y el código en pandas necesario para obtener la respuesta.

Finalmente, se implementó la interfaz tabular, que recibe la consulta del usuario, solicita al modelo los filtros correspondientes, ejecuta el código generado sobre los DataFrames y devuelve la porción filtrada de la tabla.

Implementación de la base de grafos

Los datos de relaciones entre productos se transformaron en un formato compatible con bases de grafos consultables mediante Cypher. Este proceso incluyó:

- la estandarización de identificadores y nombres de productos,
- la creación de un DataFrame con columnas específicas para nodos y relaciones,
- la preparación de los datos para su inserción en una base que soporte Cypher.

La estructura del sistema incluye la interfaz para convertir consultas en lenguaje natural a consultas Cypher mediante un LLM y la función que ejecutará la consulta Cypher sobre el grafo una vez insertados los nodos y las relaciones.

La lógica de consulta queda de esta forma establecida, permitiendo completar la etapa de inserción sin modificar la arquitectura del sistema.

Modelo de clasificación de intención

Para dirigir cada consulta hacia la fuente de datos correspondiente, se implementaron dos clasificadores de intención: uno entrenado y otro basado en un modelo de lenguaje.

El primer clasificador se entrenó con un conjunto sintético de 30 preguntas, equilibradas entre las clases *vectorial*, *tabular* y *grafo*. Se utilizó un pipeline TF-IDF con unigramas y bigramas, seguido de un modelo de Logistic Regression. El conjunto se dividió de manera estratificada y el modelo alcanzó un desempeño cercano al 0,78 de exactitud, suficiente para distinguir entre consultas semánticas, numéricas y de compatibilidad. Los errores se concentraron en ejemplos ambiguos, lo cual es razonable dado el tamaño reducido del dataset.

El segundo clasificador se implementó mediante *few-shot prompting* con Gemini. Se le proporcionaron descripciones breves de cada clase y ejemplos etiquetados. En las pruebas realizadas, este clasificador logró identificar correctamente todos los casos evaluados, mostrando mayor robustez semántica que el modelo entrenado, aunque con mayor costo y latencia.

En la integración final del asistente se habilitaron ambos métodos, permitiendo seleccionar el preferido mediante un parámetro. Por defecto se utiliza el clasificador entrenado, ya que no depende de la API externa y ofrece un rendimiento aceptable para la mayoría de las consultas.

Búsqueda híbrida y re-ranking

En la fuente vectorial se implementó un pipeline de recuperación compuesto por tres etapas: búsqueda híbrida, re-ranking y generación de respuesta.

En la primera etapa se combinan dos mecanismos complementarios. Por un lado, se utiliza BM25 para capturar coincidencias literales entre la consulta del usuario y los documentos. Por otro, se realiza una búsqueda semántica en ChromaDB empleando embeddings multilingües. Ambos resultados se unifican y normalizan, y luego se calcula un puntaje híbrido que permite ordenar los fragmentos recuperados teniendo en cuenta tanto términos exactos como similitud conceptual.

En la segunda etapa, los fragmentos mejor posicionados pasan por un proceso de re-ranking mediante un CrossEncoder. Este modelo evalúa cada par (consulta, fragmento) de forma conjunta, asignando un puntaje más preciso de relevancia. Si el re-ranking no puede ejecutarse, el sistema recurre de manera segura al orden híbrido original.

Finalmente, los fragmentos seleccionados se incorporan como contexto en el prompt enviado al LLM, que formula una respuesta basada únicamente en esa información. Este enfoque reduce la aparición de alucinaciones y mejora la pertinencia de las respuestas, especialmente en consultas en las que el usuario mezcla términos técnicos con formulaciones generales.

Pipeline de recuperación del sistema RAG

El pipeline de recuperación integra el clasificador de intención con los mecanismos específicos de cada fuente de datos. El proceso comienza cuando el usuario realiza una consulta en lenguaje natural. El clasificador determina si la pregunta corresponde a la fuente vectorial, tabular o de grafos. A partir de esta decisión, el sistema ejecuta uno de los siguientes pasos:

1. Vectorial: se realiza una búsqueda híbrida que combina BM25 y búsqueda semántica mediante embeddings. Los resultados se unifican y se reordenan utilizando un modelo CrossEncoder, que permite priorizar los fragmentos más relevantes.

2. Tabular: el sistema genera filtros dinámicos en pandas mediante un modelo de lenguaje especializado en análisis de datos. Dichos filtros se ejecutan directamente sobre los DataFrames.

3. Grafo: la consulta en lenguaje natural se transforma en una consulta Cypher, que se ejecuta sobre el grafo de productos y compatibilidades.

Una vez recuperada la información desde la fuente correspondiente, los resultados se utilizan como contexto para el modelo generativo, que formula una respuesta final en lenguaje natural basada únicamente en la evidencia disponible

Elección del LLM y entorno de ejecución

Para las tareas de comprensión de lenguaje natural, generación de código y síntesis de respuestas se utilizó un modelo de lenguaje alojado en la nube. En particular, se empleó Gemini, accedido mediante la API oficial de Google. La decisión se fundamenta en tres aspectos principales: la imposibilidad práctica de ejecutar modelos avanzados de manera local en Google Colab, la necesidad de un modelo capaz de generar código válido en español (tanto para pandas como para Cypher), y la facilidad de integración que ofrece la librería google-generativeai, que permite centralizar todas las llamadas al LLM en una única función.

Se evaluó la posibilidad de utilizar modelos locales, pero fueron descartados debido a su menor rendimiento en tareas de generación de código y a la complejidad de ejecución en entornos sin GPU dedicada. Por estos motivos, Gemini se adopta como el LLM principal para el Ejercicio 1.

Asistente conversacional integrado

Con los pipelines vectorial, tabular y de grafo implementados, se desarrolló un asistente conversacional que actúa como interfaz unificada del sistema. Se definieron dos variantes. La primera, asistente_electro, utiliza el clasificador entrenado para dirigir las consultas y aplica los pipelines básicos de cada fuente. La segunda, asistente_electro_avanzado, incorpora el pipeline vectorial mejorado (búsqueda híbrida y re-ranking) y los mismos procesos tabulares y de grafo.

Además, se implementó una función que crea un bucle conversacional con memoria. Esta función almacena las consultas y respuestas previas y permite interactuar con el asistente de forma continua, reproduciendo un escenario de uso real. Para pruebas rápidas se incluyó también un modo interactivo por consola, que finaliza cuando el usuario ingresa “EXIT” o “SALIR”.

Resultados del Ejercicio 1

Clasificador de intención

El clasificador entrenado (TF-IDF + Logistic Regression) alcanzó un rendimiento cercano al 0,78 de exactitud en un conjunto de prueba reducido, mostrando una buena capacidad para distinguir entre consultas semánticas, tabulares y de compatibilidad. El clasificador basado en LLM, evaluado con un conjunto de ejemplos, logró identificar correctamente todos los casos probados.

En la práctica, el clasificador entrenado se adoptó como opción por defecto debido a su bajo costo computacional y a su independencia de servicios externos, mientras que el clasificador LLM quedó como alternativa para escenarios en los que se prioriza la máxima precisión semántica.

Evaluación global del sistema RAG

Los resultados obtenidos muestran que los componentes del sistema interactúan de manera coherente y permiten responder una amplia variedad de consultas. La clasificación de intención orienta efectivamente cada pregunta hacia la fuente adecuada, lo que evita ejecutar procesos innecesarios o incorrectos.

El pipeline vectorial mejorado ofrece respuestas precisas incluso ante consultas ambiguas; la fuente tabular permite resolver preguntas de carácter cuantitativo sin exponer detalles técnicos; y la fuente de grafos proporciona una representación clara de compatibilidades entre productos que sería difícil de obtener desde texto o tablas.

En conjunto, el sistema cumple con los objetivos planteados para el Ejercicio 1 y establece una base para la extensión hacia un agente autónomo en el Ejercicio 2.

Imágenes de resultados

Consulta 1:
Usuario: ¿En qué provincias se concentraron más las ventas?
🔧 Código generado por el modelo: ventas['cliente_provincia'].value_counts()

Asistente (tabular): Las provincias donde se concentraron más ventas son Jujuy, Mendoza y Buenos Aires, seguidas de cerca por Neuquén y Formosa. La tabla muestra la cantidad de ventas realizadas en cada provincia, permitiendo identificar las zonas geográficas con mayor actividad comercial de electrodomésticos.

Consulta 2
Usuario: ¿Cuál es el voltaje y consumo de la licuadora?

Asistente (vectorial): La información disponible indica que tenemos diferentes modelos de licuadoras con distintos voltajes y consumos.

* **Licuadora:** Funciona con 12V y tiene un consumo de 650W.
* **Ultra Licuadora:** Funciona con 12V y tiene un consumo de 600W.
* **Compacto Licuadora:** Funciona con 220V y tiene un consumo de 1000W.
* **Licuadora:** Funciona con 110-220V y tiene un consumo de 1350W.

Para darte la información correcta, por favor, indicame el modelo exacto de tu licuadora.

Consulta 3:
Usuario: ¿Cuáles son las licuadoras de menos de $400?
🔧 Código generado por el modelo:

productos[(productos['nombre'].str.contains('Licuadora', case=False)) & (productos['precio_usd'] < 400)]

Asistente (tabular): Las siguientes licuadoras tienen un precio inferior a $400:

* Licuadora TechHome (Blanca, 283.63 USD)
* Plus Licuadora Pro TechHome (Negra, 329.07 USD)
* Compacto Licuadora ChefMaster (Rosa, 259.42 USD)

Esto significa que estas tres licuadoras son las opciones disponibles que cumplen con tu criterio de precio.

Consulta 4:
Usuario: ¿Qué productos están relacionados con la categoría Cocina?”
🔧 Cypher generado por el modelo:

MATCH (p:Producto)-[r:COMPATIBLE_CON]->(c:Producto) WHERE p.categoria = "Cocina" RETURN c.id AS id_producto, c.nombre AS nombre, c.categoria AS categoria, c.marca AS marca, r.comparte AS componente_compartido

Asistente (grafo): Los productos relacionados con la categoría "Cocina" son:

* **Marca KitchenPro:** Advanced Batidora de Pie, Profesional Abridor de Latas, Turbo Exprimidor, Olla de Cocción Lenta, Turbo Microondas, Plus Parrilla Eléctrica, Turbo Abridor de Latas, Deluxe Yogurtera 3000, Olla de Cocción Lenta X, Smart Conservadora 2024, Procesadora, Deluxe Conservadora Pro, Max Pava Eléctrica, Abridor de Latas, Procesadora y Mixer. Algunos comparten componentes como "Accesorios", "Jarra", "Panel de Control", "Cuchillas" o "Motor".

* **Marca CookElite:** Eco Mixer II, Deluxe Abridor de Latas, Olla de Cocción Lenta II, Sandwichera, Pava Eléctrica, Profesional Yogurtera, Elite Heladera, Max Conservadora, Tostadora, Advanced Freezer, Vinoteca Pro, Smart Yogurtera, Parrilla Eléctrica, Freezer, Plus Olla de Cocción Lenta, Abridor de Latas 2024, Waflera, Digital Microondas, Elite Deshidratador, Premium Heladera, Digital Olla de Cocción Lenta. Algunos comparten componentes como "Motor", "Jarra", "Accesorios", "Panel de Control", "Cuchillas"

*   **Marca HomeChef:** Frigobar, Eco Waflera, Digital Sandwichera Plus, Profesional Procesadora 2024, Abridor de Latas, Batidora de Pie II, Premium Molinillo de Café, Vinoteca II, Pava Eléctrica 2024, Turbo Conservadora, Premium Cafetera, Olla de Cocción Lenta Plus, Mixer II, Batidora de Mano II, Advanced Yogurtera, Sandwichera 3000, Pro Vinoteca, Pro Waflera, Digital Olla de Cocción Lenta, Mixer Pro, Algunos comparten componentes como "Accesorios", "Jarra", "Panel de Control", "Cuchillas" o "Motor".

*   **Marca TechHome:** Compacto Horno Eléctrico, Vinoteca, Licuadora, Max Molinillo de Café, Freidora de Aire, Profesional Mixer, Olla de Cocción Lenta, Deluxe Procesadora, Ultra Parrilla Eléctrica, Profesional Freezer, Digital Rallador Eléctrico, Turbo Deshidratador, Yogurtera, Ultra Licuadora, Super Freezer, Elite Tostadora, Frigobar, Eco Rallador Eléctrico. Algunos comparten componentes como "Panel de Control", "Jarra", "Cuchillas", "Accesorios" o "Motor".

*   **Marca ChefMaster:** Eco Conservadora II, Deluxe Exprimidor X, Advanced Heladera, Deshidratador, Waflera, Sandwichera 3000, Pro Vinoteca, Pro Waflera, Digital Olla de Cocción Lenta, Mixer Pro, Compacto Rallador Eléctrico 2024, Batidora de Mano, Licuadora. Algunos comparten componentes como "Accesorios", "Panel de Control", o "Cuchillas".

Es importante destacar que los productos de diferentes marcas pueden compartir componentes entre sí.

Consulta 5:
Usuario: ¿cuales son las licuadoras disponibles?
🔧 Código generado por el modelo:

```
productos[productos['nombre'].str.contains('Licuadora', case=False)]
```

Asistente (tabular): Las licuadoras disponibles son:

*   Licuadora TechHome (P0001)
*   Licuadora TechHome (P0002)
*   Plus Licuadora Pro TechHome (P0003)
*   Compacto Licuadora ChefMaster (P0004)
*   Licuadora HomeChef (P0005)
*   Licuadora ChefMaster (P0006)
*   Licuadora KitchenPro (P0007)
*   Ultra Licuadora TechHome (P0008)

La tabla muestra el detalle de cada licuadora, incluyendo su marca, precio, stock, color, potencia, capacidad y otras características relevantes.

Mejoras propuestas

En la práctica, se observaron algunas oportunidades de mejora vinculadas principalmente a la recuperación de información en los manuales. Si bien el tamaño de fragmento elegido (512 caracteres con solapamiento de 64) resultó razonable para unificar el tratamiento de todo el corpus, sería interesante en trabajos futuros probar otros tamaños y solapamientos, especialmente en manuales extensos, así como explorar una segmentación más guiada por la estructura del documento (por ejemplo, encabezados o secciones).

Conclusiones

El sistema desarrollado en este trabajo permitió integrar de manera efectiva distintas fuentes de información para responder consultas en lenguaje natural. La combinación de recuperación semántica, consultas tabulares generadas por un modelo y la generación de consultas Cypher demostró que es posible construir un asistente versátil dentro del dominio de electrodomésticos. El desempeño del clasificador de intención y del pipeline vectorial mostró ser adecuado para esta etapa, y el sistema respondió de forma consistente a diferentes tipos de preguntas.

Bibliografía

Fuentes de datos: https://drive.google.com/drive/folders/12FNaPe1Mqs3QQ8QBp78WUEXgTVKnVbeK?usp=sharing

ChromaDB - Open-source Vector Database: https://www.trychroma.com/

Neo4j Cypher Query Language: https://neo4j.com/docs/cypher-refcard/current/

Gemini Model by Google: https://ai.google.dev/api?hl=es-419&lang=python