

*Maestría en Ciencia de Datos (Virtual)*

*Facultad de Ingeniería*

**Trabajo Práctico – Text Mining**

**Desarrollo de un Asistente Farmacológico basado en Recuperación Aumentada por Generación (RAG)**

**Docente:**

* Hernan Merlino

**Integrantes del Grupo:**

* Brusasca, Lucas
* Durán, Pedro
* Gaddi, Martín
* Lijtmaer, Paul
* Palavecino, Nicolás

**Fecha de entrega:** 15 de Julio 2025

### 1. Introducción

El acceso confiable, actualizado y comprensible a la información sobre medicamentos es fundamental en contextos clínicos, educativos y de investigación. En este trabajo se desarrolla un asistente farmacológico inteligente utilizando la arquitectura RAG (Retrieval-Augmented Generation), que combina modelos de lenguaje (LLMs) con recuperación semántica para responder preguntas relacionadas con más de 17.000 fármacos extraídos desde [Drugs.com](https://www.drugs.com/).

Se abordan todas las etapas necesarias: recolección (*scrapping*) de los datos, procesamiento, vectorización, indexación semántica, generación de respuestas y construcción de una aplicación interactiva.

### 2. Recolección de Datos *(Scrapping links.ipynb y Scrapping texts v2.ipynb)*

La construcción del corpus comenzó con la automatización del scraping de Drugs.com. Primero, se extrajeron los enlaces individuales de cada medicamento utilizando Selenium y BeautifulSoup en el notebook **Scrapping links.ipynb**. Esta etapa generó una base con más de 17.000 URLs. Por la composición de la página, debió realizarse una búsqueda por cada par de letras iniciales (así se generan las urls en el buscador). A su vez, se debió considerar los “bloqueos” que la página generaba a consultas reiteradas, a través de pausas randomizadas en su ejecución.

A partir de estos enlaces, **Scrapping texts v2.ipynb** accedió a cada página de forma individual para extraer su contenido textual. El HTML fue parseado cuidadosamente para capturar tanto la jerarquía estructural de la información (usando etiquetas como <h2>, <p>, <ul> y <ol>) como para preservar la semántica original. Esta jerarquización fue clave para permitir posteriormente una segmentación precisa en documentos autocontenidos. Esta etapa tuvo el mismo desafío que la búsqueda de las url.

Dada la magnitud del corpus y la alta tasa de errores por timeouts o rechazos del servidor, se optó por paralelizar la ejecución. Se separaron los datos en bloques aproximadamente iguales, permitiendo correr el scraping en paralelo en distintas máquinas o hilos, unificándolos al final. El proceso completo tomó varios días de ejecución. El peso de los documentos, llegó a más de 500 MB combinados.

### 3. Procesamiento y Construcción de Documentos *(Armado de documentos.ipynb)*

Los textos recolectados contenían una estructura jerárquica compleja. Se desarrolló una función que analizaba la jerarquía HTML para identificar títulos (<h2>) y su contenido asociado, agrupándolo en bloques de texto autocontenidos. Esto permitió dividir un artículo completo sobre una droga en varios “documentos” individuales, cada uno representando una sección temática como “What is…”, “Side Effects”, “Dosage”, etc. Esto se hizo con el objetivo de brindar al modelo, información mejor compartimentalizada.

Cada bloque de texto fue precedido por el nombre del medicamento para mantener el contexto incluso si el chunk se encuentra aislado. El resultado final fue un archivo “**.jsonl**” de ~300 MB con entradas de la forma:

A white background with red text

AI-generated content may be incorrect.

Esta estructura es ideal para sistemas RAG que requieren unidades documentales compactas y semánticamente coherentes.

### 4. Construcción del Sistema RAG (Embeddings e Indexado)

Una vez conformado el corpus, se procedió a construir el sistema RAG. Se utilizaron **sentence-transformers** para generar embeddings semánticos de cada documento. Estos vectores fueron almacenados en un índice FAISS (**index\_recursive.faiss**), optimizado para búsquedas por similitud cóseno.

Para preparar los textos, se empleó **RecursiveCharacterTextSplitter** de **langchain.text\_splitters**, configurado con chunks de 1000 caracteres y 100 de solapamiento (habiendo probado con varias alternativas, hasta decantar por esta por sus resultados satisfactorios). Esta estrategia garantiza que no se corten conceptos críticos entre documentos consecutivos.

Con esto, cualquier pregunta del usuario puede ser vectorizada, y sus vecinos más cercanos en el espacio FAISS se recuperan para enriquecer el contexto del modelo generador.

### 5. Implementación del Asistente Farmacológico *(asistente\_farmacologico.py)*

La interfaz principal del sistema está implementada en **asistente\_farmacologico.py**, usando:

* **LangChain** como marco de orquestación.
* **FAISS** para la búsqueda semántica eficiente.
* **Deep Translator** para detección y traducción automática de idioma.
* **Ollama** como backend para LLMs, en este caso usando llama3 como modelo local.

El pipeline sigue este flujo:

1. Se escribe la pregunta (en inglés u español).
2. Se detecta si la pregunta está en inglés o español.
3. Traduce a inglés si es necesario (lenguaje de los datos de origen).
4. Recupera los documentos relevantes desde FAISS.
5. Arma un prompt con contexto + pregunta.
6. Genera una respuesta usando LLM (con el modelo seleccionado, por default llama 3.1)
7. Traduce al idioma deseado (seleccionado en el inicio)

Esta arquitectura garantiza respuestas personalizadas, centradas en el medicamento consultado, con buena fluidez y precisión.

### 6. Evaluación y Desafíos

Se evaluó el sistema con consultas variadas como:

* *What is Abacavir?*
* *What are the side effects of Ozempic?*
* *Should I take Ibuprofen when pregnant?*
* *Etc.*

Los resultados fueron satisfactorios: respuestas concisas, sin repeticiones, y con redacción natural. Se observaron mejoras al utilizar modelos LLM de mayor capacidad como **llama3**, frente a **distilgpt2**, especialmente en consultas específicas.

Sin embargo, se detectaron algunos desafíos pendientes:

* **Tiempo de respuesta**: actualmente cuando el sistema se ejecuta sin GPU, no tiene una respuesta demasiado rápida.
* **Gestón de contexto**: en preguntas muy generales, la calidad baja por falta de enfoque. Mejorar el sistema de prompt para reforzar la relevancia.
* **Faltan fuentes externas**: el corpus está limitado a Drugs.com. Podrían Incorporarse otras fuentes como MedlinePlus o la FDA para mejorar la cobertura.

### 7. Conclusiones

Este proyecto demuestra la viabilidad de construir un asistente farmacológico local basado en RAG, combinando scraping web, procesamiento estructurado de texto, embeddings y modelos generativos.

El sistema es escalable, funcional offline y respetuoso de la privacidad. Su aplicación puede extenderse a:

* Plataformas educativas y capacitaciones sanitarias.
* Sistemas de apoyo a farmacias y profesionales.
* Consulta de usuarios de drogas, con dudas sobre sus riesgos.

Con mejoras adicionales en rendimiento y cobertura, este asistente podría convertirse en una herramienta robusta de consulta sanitaria.

### 8. Anexos

* Disponibilizamos en: xxxxxxxxxx los archivos con sus datos. Por peso de los archivos (la carpeta completa pesa varios gigas), no se pueden entregar en directa. Para ejecutarlo, la carpeta debe verse así:

A su vez, se sugiere generar un entorno nuevo con los requerimientos detallados en “requirements.txt”, para asegurar replicabilidad.

* Se adjunta en el mail un demo para mostrar la funcionalidad sin necesidad de correrlo localmente.