UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

T.U.I.A

PROCESAMIENTO DE LENGUAJE NATURAL

TRABAJO PRÁCTICO 2

2024

Alumno:

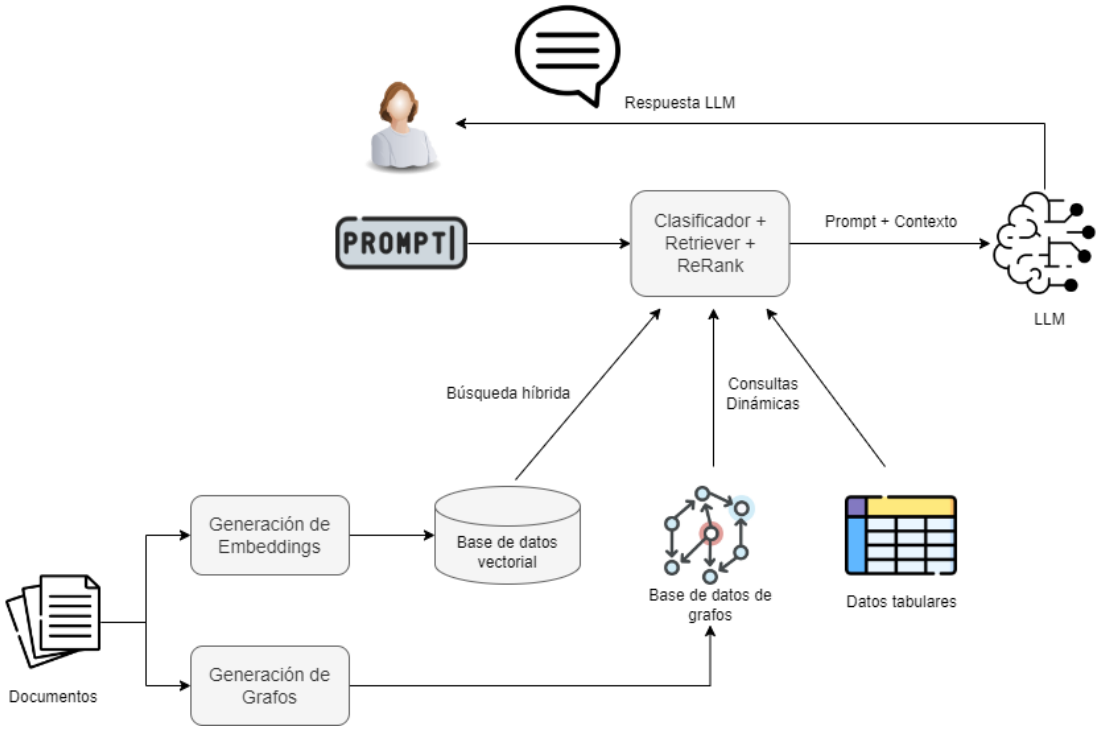
Pistelli, Pablo

ÍNDICE

EJERCICIO 1 - DESCRIPCIÓN DEL TRABAJO PRÁCTICO

La primera parte de este trabajo consiste en desarrollar un Chatbot experto en el juego de mesa "Viticulture" utilizando la técnica RAG (Retrieval Augmented Generation).

El objetivo es que el sistema pueda entablar un diálogo con el usuario tanto en español como en inglés y pueda responder las preguntas sobre el juego recurriendo a diferentes fuentes de datos provistas.



FUENTES DE DATOS

El primer paso fue buscar información sobre el juego y obtener archivos y fuentes para extraerla, procesarla e ingresarla a nuestro sistema

BASES DE DATOS DE TEXTO

01_EXTRACCION_TEXTOS.IPYNB

Mediante Web Scrapping:

- FAQs de la página oficial del desarrollador del juego:
    - https://stonemaiergames.com/games/viticulture/faq/

- Reseñas en diferentes páginas:
    - https://collectible506.com/2020/09/10/una-grandiosa-resena-para-un-hermoso-juego-viticulture-essential-edition-stonemaier-games/
    - https://misutmeeple.com/2015/03/resena-viticulture/
    - https://www.darkstone.es/resenas-y-entrevistas-darkstone/resena-darkstone-viticulture-essential-edition/?PHPSESSID=260664c0d841412d5e2d21f46218b04f
    - https://elusivemeeple.com/2018/05/27/viticulture-review/

- Artículos sobre estrategias para el juego:
    - https://elusivemeeple.com/2018/05/27/viticulture-strategy-tips/
    - https://www.cardboardrepublic.com/spotlight-material/official-strategy-guide-for-viticulture

Desde archivos PDF:

- Manuales oficiales de la página oficial del juego
    - https://stonemaiergames.com/games/viticulture/rules/
    - https://www.dropbox.com/scl/fo/71ou252te9g6kfz25biz9/AHLQPkHwxFnYxXYROVh1mws?rlkey=nud55i7z7tvgivnweu43jluyl&e=1&dl=0
    - https://boardgamegeek.com/boardgame/128621/viticulture/files?pageid=1

Desde videos de YouTube:

- Reglas y mecánicas del juego
    - https://www.youtube.com/watch?v=cFNdH88bTKs
    - https://www.youtube.com/watch?v=Pde-3rJbbHc
    - https://www.youtube.com/watch?v=xRkt0aamraY

Desde archivos de audio:

- Podcast sobre el juego
    - https://boardsalivepodcast.com/2015/12/21/episode-41-viticulture-essential-edition-ba-holiday-special/

Todas las fuentes fueron convertidas o transcriptas a texto plano utilizando las herramientas provistas en la materia, reutilizando el código brindado en la teoría y parte del código desarrollado para el TP1.

Una primera cuestión a definir fue el idioma de la base de datos. Algunas de las fuentes se encontraban en español y otras en inglés. Si bien las herramientas y modelos tienen mayor desarrollo para trabajar en inglés, decidí trabajar las bases de datos en español para poder probar los modelos multilenguaje. Un buen ejercicio sería repetir el trabajo con las bases de datos en inglés y poder hacer una comparación del rendimiento.

BASES DE DATOS TABULARES

02_DATOS_TABULARES.IPYNB

Para las bases de datos tabulares trabajé con dos archivos descargados de BGG:

- Ranking de juegos de Octubre 2024
    - https://bgg.activityclub.org/bggdata/bgg-ranking-historicals/2024-10-25T00-36-11.csv

- Listado de cartas de visitantes del juego
    - https://boardgamegeek.com/file/download_redirect/f797105f69b033a81e25da86d31ab48923db48db3a6e5b91/French+Visitors+cards.xlsx

- Cartas de ordenes de vinos (sólo analicé el contenido y utilicé los datos para generar un grafo)
    - https://boardgamegeek.com/file/download_redirect/73e997b0eec72300242dc1ee245983c24168a89626169822/Vine+order+cards+%28purple%29.xlsx

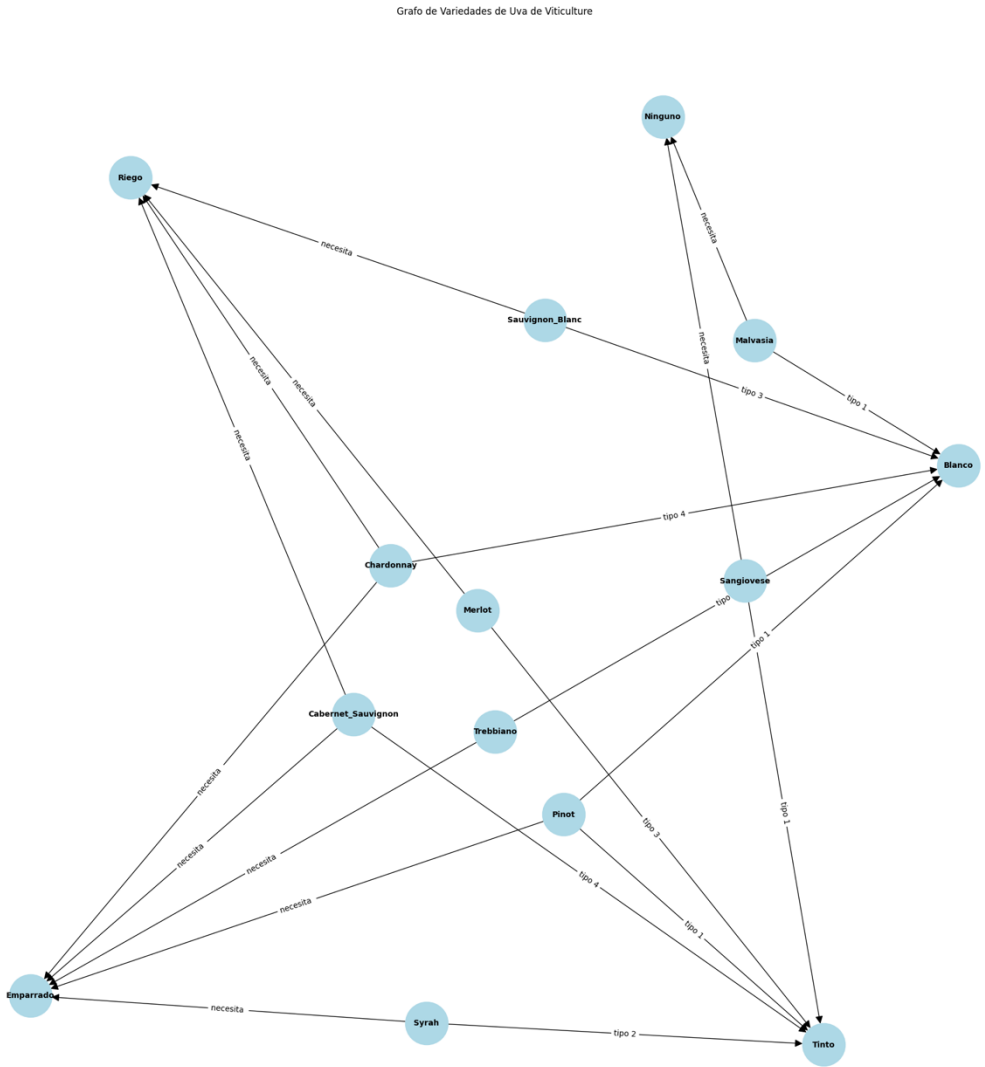
BASES DE DATOS DE GRAFOS

03_GRAFOS.IPYNB

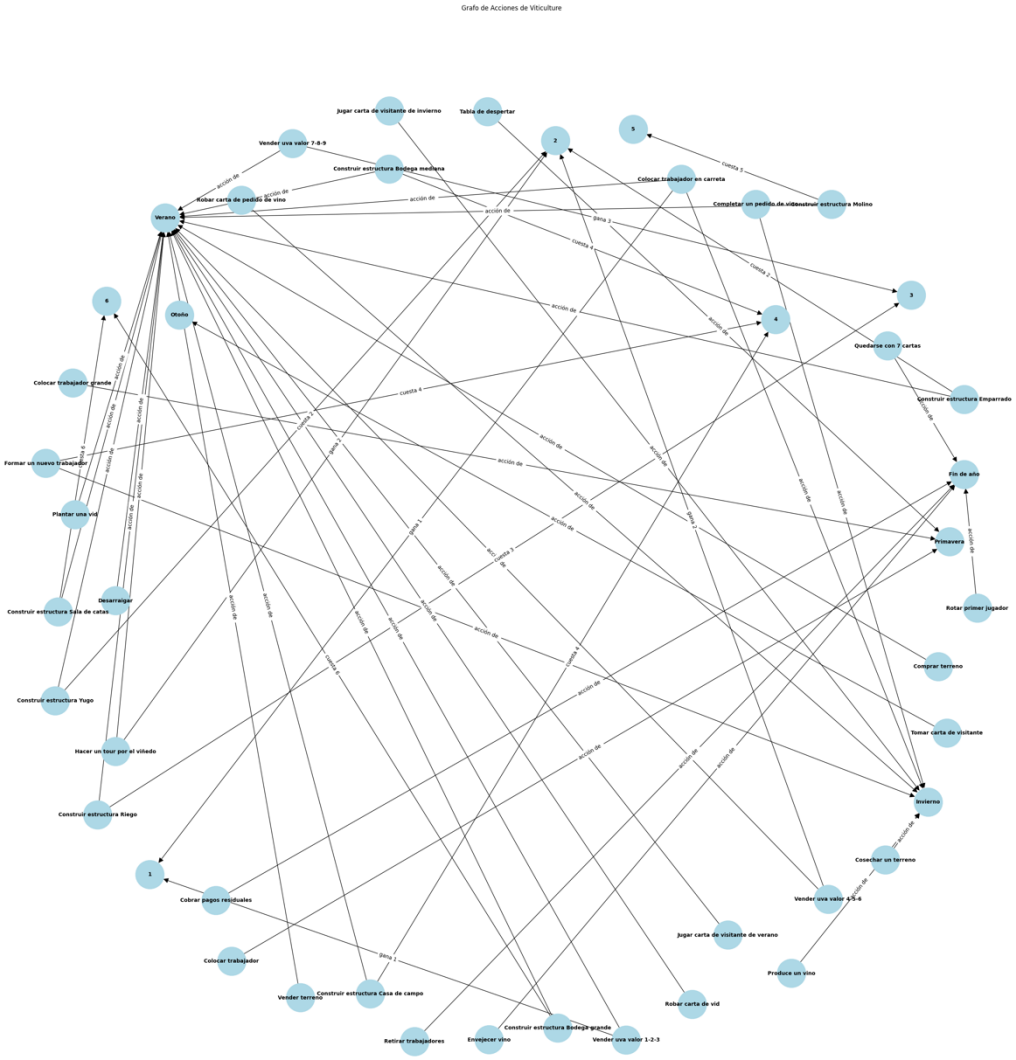
Al no contar con bases de datos de grafos en fuentes online como Wikidata, necesité generar mis propios grafos a partir de información recopilada sobre el juego.

Utilizando archivos descargados desde el sitio BoardGameGeek, generé los siguientes grafos manualmente utilizando la herramienta *networkx*:

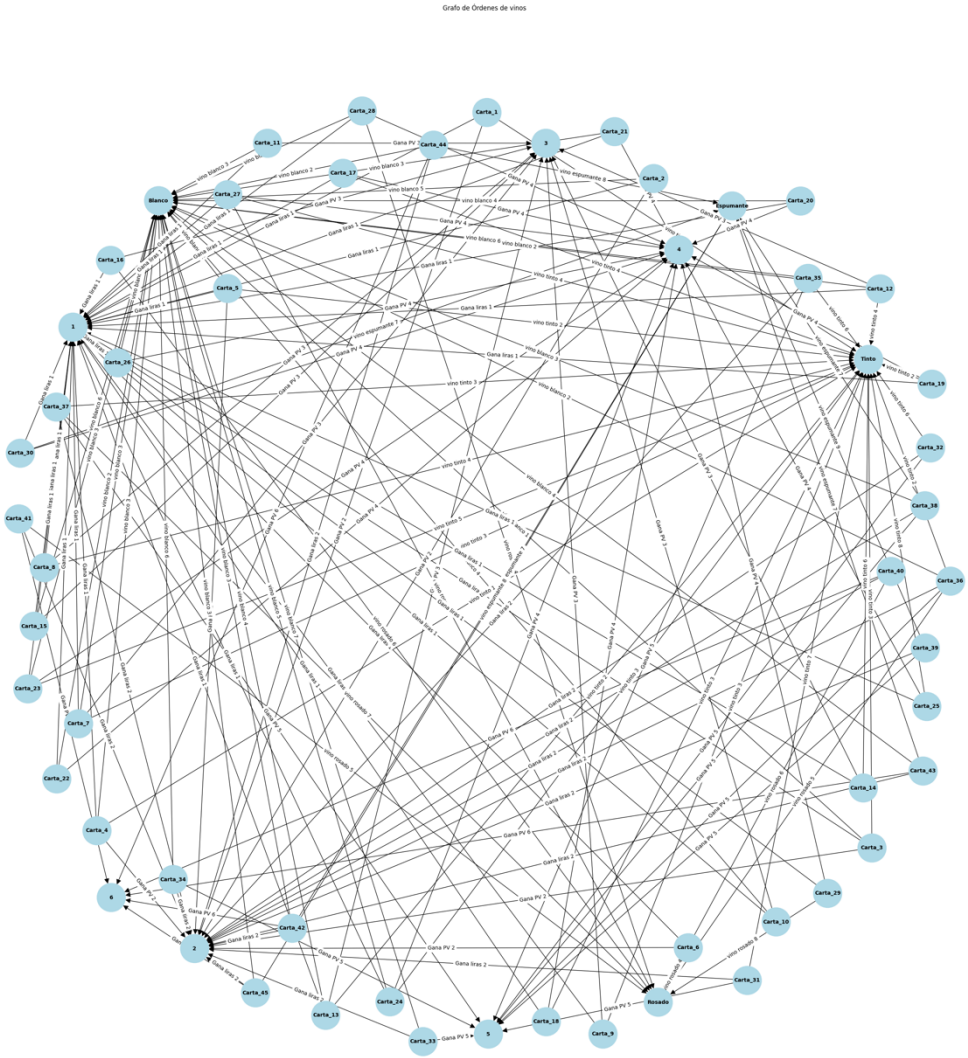
- "Características según los tipos de uva": a partir de una tabla extraída del manual oficial



- “Acciones”: también a partir del manual original

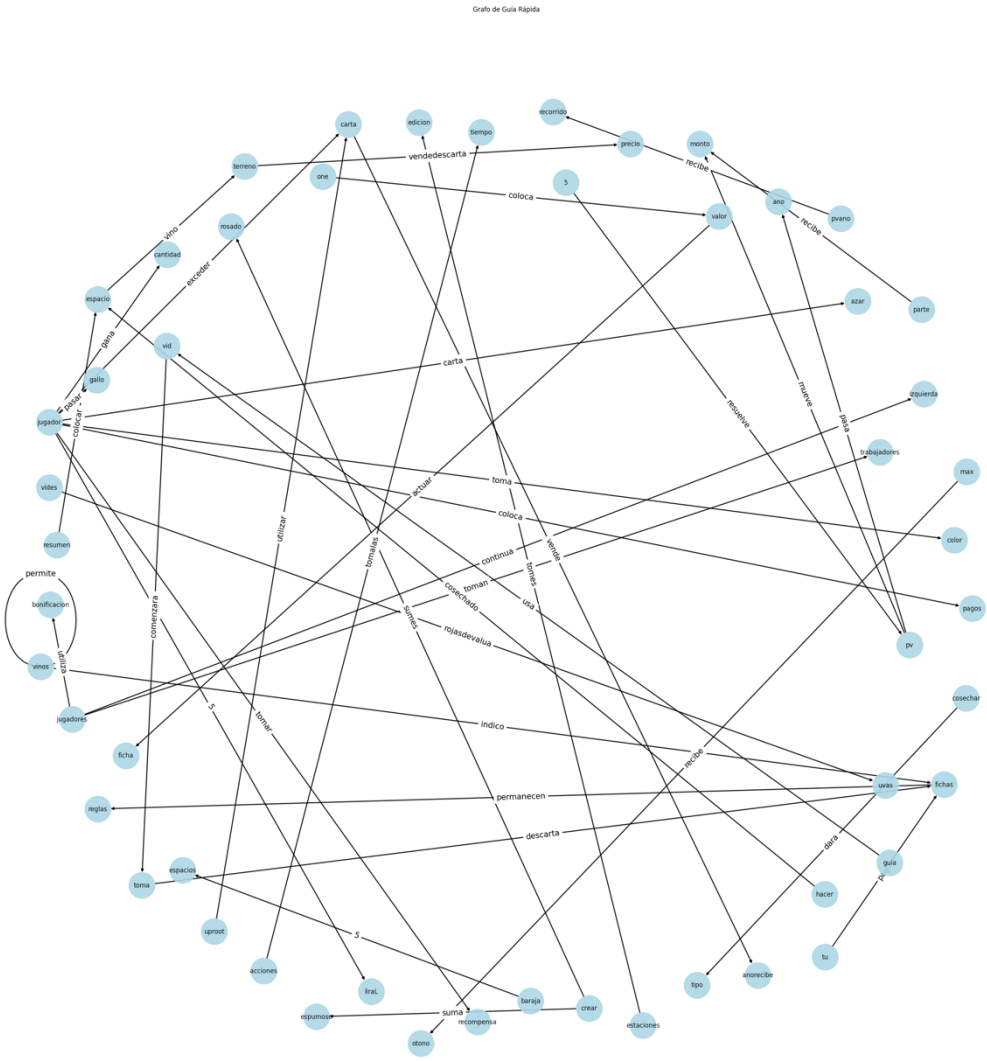


- “Cartas de órdenes de vinos”: a partir de un archivo xlsx descargado de BGG



Además, a partir de una guía rápida del juego (un resumen del manual), generé un grafo utilizando técnicas de procesamiento de texto y luego POS.

El resultado no fue satisfactorio, ya que a simple vista, las tríadas generadas no aportan información útil. Luego de leer completo el texto original, creo que se debe al contenido y tipo de redacción de la guía rápida.



CLASIFICADOR DE CONSULTAS

04_CLASIFICADORES.IPYNB

El siguiente paso es crear un clasificador para definir de qué fuente de datos buscar la información solicitada. La consigna indica utilizar dos clasificadores y compararlos, uno basado en embeddings y otro en un modelo LLM.

Según las fuentes de datos con las que contaba definí las siguientes categorías:

- "Reglas del juego" (textos en base de datos vectorial)
- "Mecánicas del juego" (textos en base de datos vectorial)
- "Historia del juego" (textos en base de datos vectorial)
- "Estrategias" (textos en base de datos vectorial)
- "Tipos de uvas" (grafo de uvas)
- "Acciones que se pueden realizar" (grafo de acciones)
- "Cartas de órdenes de vinos" (grafo de órdenes de vinos)
- "Cartas de visitantes" (tabla de visitantes)
- "Ranking" (tabla de ranking)

Para el clasificador basado en embeddings utilicé un modelo de **regresión logística** aplicando el código provisto en la Unidad 3. Entrenado con varios ejemplos para cada categoría los resultados fueron aceptables.

```
# Ejemplo
consulta = "¿Qué pasa si no puedo cumplir una carta de pedido de vino?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta(consulta))
```

```
Consulta: ¿Qué pasa si no puedo cumplir una carta de pedido de vino?
Categoría predicha: estrategias
```

```
# Ejemplo
consulta = "¿Quién es el creador del juego?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta(consulta))
```

```
Consulta: ¿Quién es el creador del juego?
Categoría predicha: historia del juego
```

```
# Ejemplo
consulta = "¿Qué hace la carta de visitante Proveedor?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta(consulta))
```

```
Consulta: ¿Qué hace la carta de visitante Proveedor?
Categoría predicha: cartas de visitantes
```

Una mejora posible en este punto sería aumentar la cantidad de datos de entrenamiento.

Para el clasificador basado en un modelo LLM se utilizó **GPT2**. En este caso fue un poco más complejo de ajustar, y si bien esperaba los mejores resultados con este clasificador, no se lograron buenos resultados.

```
# Ejemplo
consulta = "¿Qué pasa si no puedo cumplir una carta de pedido de vino?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_gpt(consulta))
```

```
Consulta: ¿Qué pasa si no puedo cumplir una carta de pedido de vino?
Categoría predicha: Reglas del juego
```

```
# Ejemplo
consulta = "¿Cuándo se creó el juego?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_gpt(consulta))
```

```
Consulta: ¿Cuándo se creó el juego?
Categoría predicha: Reglas del juego
```

```
# Ejemplo
consulta = "¿Qué hace la carta de visitante Proveedor?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_gpt(consulta))
```

```
Consulta: ¿Qué hace la carta de visitante Proveedor?
Categoría predicha: Reglas del juego
```

Al no obtener buenos resultados, se generó un nuevo clasificador utilizando la API de **HuggingFace** con el modelo [HuggingFaceH4/zephyr-7b-beta](). Si bien el código resultó más complejo de imlpementar, el resultado obtenido fue notablemente mejor.

```
    # Preparamos el prompt para la clasificación
    chat_prompt = [
        {"role": "system", "content": "Eres un clasificador de preguntas sobre el juego Viticulture."},
        {"role": "user", "content": f"""
        Devuelve sólo el nombre de la categoría en la que se encuentra esta pregunta:'{consulta}'.
        Las opciones son:
        Reglas del juego,
        Mecánicas del juego,
        Historia del juego,
        Tipos de uvas,
        Acciones,
        Cartas de órdenes de vinos,
        Cartas de visitantes,
        Ranking
        """}
    ]
```

```
# Ejemplo
consulta = "¿Qué pasa si no puedo cumplir una carta de pedido de vino?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_hf(consulta))
```
```
Consulta: ¿Qué pasa si no puedo cumplir una carta de pedido de vino?
Categoría predicha: Cartas de órdenes de vinos
```

```
# Ejemplo
consulta = "¿Quién es el creador del juego?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_hf(consulta))
```
```
Consulta: ¿Quién es el creador del juego?
Categoría predicha: Historia del juego
```

```
# Ejemplo
consulta = "¿Qué hace la carta de visitante Proveedor?"
print("Consulta:", consulta)
print("Categoría predicha:", clasificar_consulta_hf(consulta))
```
```
Consulta: ¿Qué hace la carta de visitante Proveedor?
Categoría predicha: Cartas de visitantes
```

De los tres clasificadores probados, se utilizará el último para continuar con el proyecto.

DESARROLLO

05_IMPLEMENTACION.IPYNB

A partir de lo desarrollado en los notebooks anteriores de implementa el Chatbot especializado en Viticulture.

En primer lugar se generan las bases de datos a partir de los documentos analizados en la primera parte.

DOCUMENTOS DE TEXTO

Primero se realiza una limpieza y acondicionamiento de los textos extraídos en la primera parte. Luego se realiza un split de los textos para posteriormente generar los embeddings de cada uno e ingresarlos en una colección de **ChromaDB**. En un principio se generó con segmentos de 1000 caracteres pero dada la extensión de los documentos se consideró que 500 era suficiente.

GRAFOS

Para incorporar los grafos se utilizó **rdflib** y se importaron los archivos .RDF generados. Se realizaron diferentes modelos de consulta **SPARQL** según lo ingresado por el usuario y se adaptó la salida para brindar un texto que pueda usarse como contexto.

TABLAS

Se importaron las tablas guardadas en .csv usando DataFrames de **pandas**. En este punto el desarrollo podría completarse aplicando técnicas del TP1 para realizar búsquedas dentro de las tablas que puedan utilizarse como contexto. Para simplificar este punto se utilizó una sola tabla completa (la de menor extensión) como contexto para el LLM.

CLASIFICADOR

Se utilizó el clasificador de mejores resultados de la parte 4 utilizando la API de **HuggingFace** con el modelo HuggingFaceH4/zephyr-7b-beta.

RERANK

Se implementó un método de ReRank que luego se utilizará sobre los resultados obtenidos de la consulta a la base de datos vectorial. Se utilizó el código provisto por la teoría usando la librería **FlagEmbedding.**

SELECCIÓN DE FUENTE

Se definió una función que, a partir de la categoría indicada por el clasificador, defina en qué base de datos realizar la consulta. Esta función generará el contexto final que se enviará junto con la consulta al modelo LLM.

GENERACIÓN DE RESPUESTAS UTILIZANDO LLM

Finalmente, con la consulta del usuario y el contexto extraído de las bases de datos, se implementa el RAG utilizando el modelo **HuggingFaceH4/zephyr-7b-beta**.

El promp generado para ingresar al modelo es del tipo:

```
    # Preparamos el prompt para la generación con el contexto
    chat_prompt = [
        {"role": "system", "content": "Eres un asistente experto en el juego Viticulture. Para responder las consultas debes tener en cuenta el contexto relevante provisto. Siempre responde en español."},
        {"role": "user", "content": f"¿{consulta}?"},
        {"role": "system", "content": f"Contexto relevante:\n{contexto}"}
    ]
```

Se realizaron algunas consultas directamente al generador de respuestas, primero sin pasar por el clasificador (eligiendo manualmente la base de datos a consultar) y luego con el sistema completo (dejando que el clasificador determine dónde buscar la información).

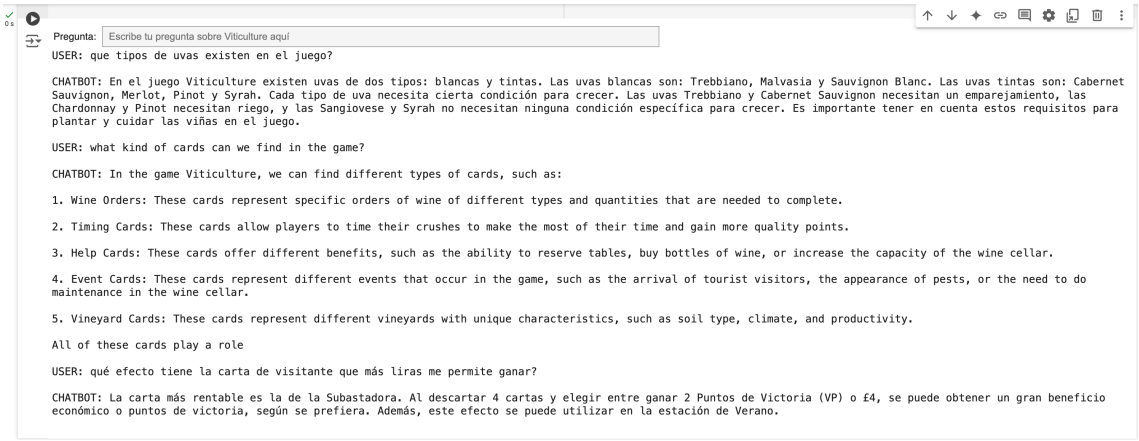
IDIOMAS

El procesamiento de los datos es enteramente en español. En caso de que la consulta se realice en inglés, primero se traduce al español y luego se ingresa al sistema. Antes de devolver al usuario la respuesta, se vuelve a traducir al inglés. Para esto volví a utilizar la librería **deep-translator** como en el TP1.

INTERACCIÓN CON EL USUARIO

Como primera opción se optó por volver a utilizar la interfaz aplicada en el TP1 con **ipywidgets**. Se generó una interfaz simple que permite el ingreso de texto en un campo y un botón para confirmar y solicitar la respuesta.

No es una interfaz óptima para la interacción con un sistema de chat, pero no pude completar la implementación de una interfaz más apropiada como **streamlit**.



MEJORAS POSIBLES

En base a lo aprendido durante el desarrollo de este proyecto identifico algunos puntos que podría mejorar:

1. Si bien se trató de generar partes compactas de código en diferentes notebooks, se podría mejorar aún más generando las bases de datos en un código individual, exportándolas en un formato almacenable e importándolas en el modelo principal. Esto simplificaría la ejecución de los notebooks.

2. Los grafos obtenidos de manera automática utilizando POS no resultan del todo satisfactorios. Se podría mejorar el preprocesamiento de los textos utilizados para lograr mejores tríadas.

3. La comparación entre clasificadores se realizó mediante un análisis subjetivo en base a las respuestas frente a mismas consultas. Podría definirse alguna métrica o método de evaluación objetivo para seleccionar cuál utilizar.

4. Si bien en este trabajo y en el anterior se desarrollaron métodos de extracción de información para todos los tipos de fuentes (tablas, grafos y texto), no se llegó a implementar la totalidad de las fuentes de datos recopiladas.

5. Podría lograrse una interfaz de usuario más dinámica.

6. En este trabajo no pude explorar en profundidad y probar otros modelos preentrenados para las diferentes aplicaciones. Esto podría optimizar los resultados obtenidos.

CONCLUSIONES

En este trabajo la interacción entre los modelos fue en ocasiones compleja de adecuar. Los resultados finales obtenidos creo que son satisfactorios aunque presentan muchas oportunidades de mejora.

Los modelos de lenguaje utilizados son abiertos y aun así, luego de varias pruebas y ajustes en los prompts, presentaron buenos resultados. Supongo que teniendo la posibilidad de usar modelos por suscripción como GPT4 representaría una mejora considerable en la respuesta a los prompts.

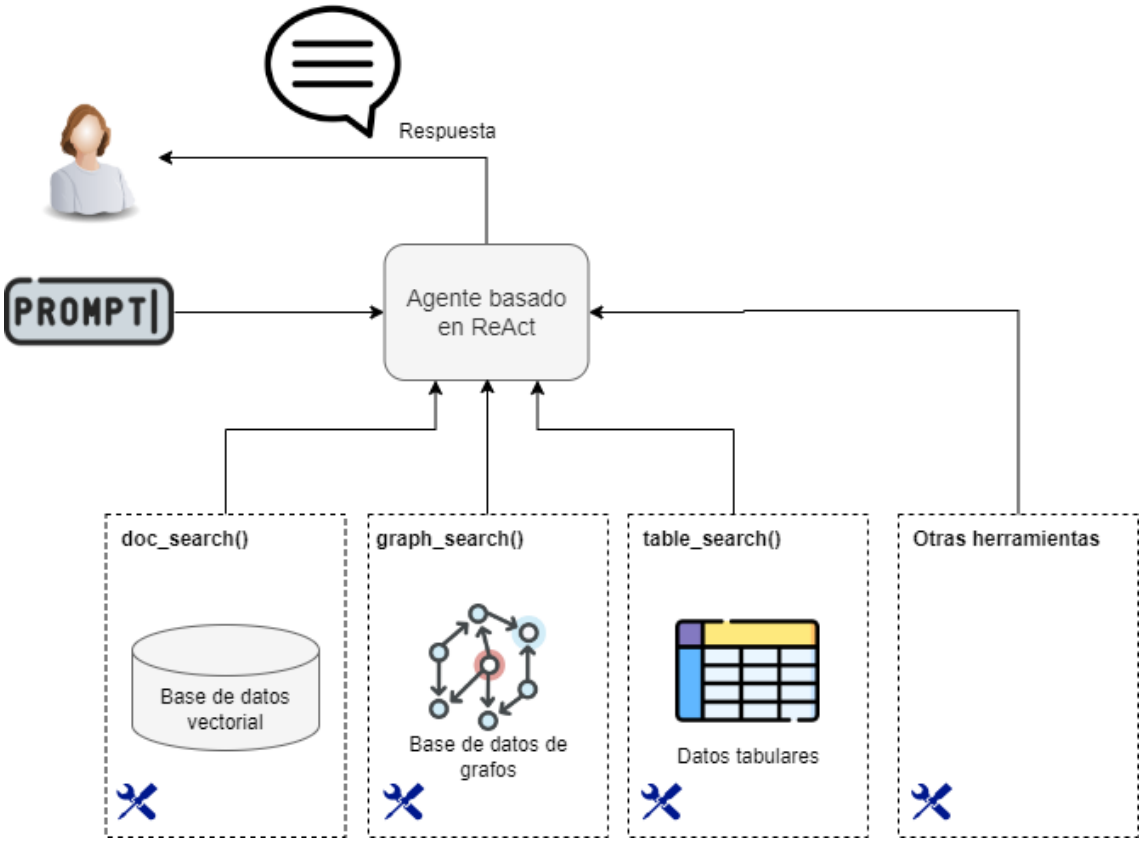
Luego de la interacción con las distintas bases de datos, fue evidente la efectividad en las consultas a las bases de grafos. Aunque reconozco que presenta una dificultad mayor al momento de estructurar las queries y en desarrollar los grafos.

El resultado de interacción multilenguaje me parece un punto importante para cualquier implementación de chatbot y no requiere de demasiado código adicional, aunque sí de procesamiento. Al realizar las preguntas en inglés noto una demora mucho mayor en la respuesta.

EJERCICIO 2 - DESCRIPCIÓN DEL TRABAJO PRÁCTICO

Este ejercicio se basa en el ejercicio 1, e incorpora el concepto de Agente, basado en el concepto ReAct. Nuestro agente debe cumplir con los siguientes requisitos:

- Utilizar al menos 3 herramientas, aprovechando el trabajo anterior:
    - doc_search(): Busca información en los documentos
    - graph_search(): Busca información en la base de datos de grafos
    - table_search(): Busca información sobre los datos tabulares
- Se puede implementar alguna nueva herramienta que se considere necesaria y que pueda enriquecer las capacidades del agente.
- Utilizar la librería Llama-Index para desarrollar el agente:
    - llama_index.core.agent.ReActAgent
    - llama_index.core.tools.FunctionTool
- Se debe construir el prompt adecuado para incorporar las herramientas al agente ReAct



DESARROLLO

06_AGENTE.IPYNB

Para el desarrollo del agente se utilizaron las herramientas mencionadas en el enunciado y provistas en el notebook de la Unidad 7.

El desarrollo de las herramientas de búsqueda está basado en las funciones utilizadas previamente para realizar las consultas a las diferentes bases de datos generadas en el proyecto.

Para evitar conflictos con el ejercicio anterior, se repitió la primera parte del notebook donde se generan las bases de datos y las funciones y se incorporó al final el agente utilizando **Llamaindex**.

EJEMPLOS DE CONSULTAS

Los resultados completos se dejan en el notebook.

PRIMER EJEMPLO

```
    queries = [
        "¿Cuál es la carta de visitante que más liras me otorga?",
        "¿De qué se trata el juego Viticulture?",
        "¿Cómo se define quién comienza la partida?",
        "¿Qué tipos de uvas existen en el juego?",
        "¿Hay alguna forma de ganar siempre?"
    ]
```

Se observa que sólo recurre a las funciones para la primera pregunta, el resto indica que puede resolverlo sin ayuda.

SEGUNDO EJEMPLO

```
    queries = [
        "¿Puedes darme información detallada de cada variedad de
uva del juego? Puntajes, tipos, etc",
        "¿Cuáles son las acciones que se pueden realizar en cada
estación?",
        "¿Puedes detallarme los elementos que componen el juego?
Cantidad de cartas, fichas, tableros, etc.",
    ]
```

En este caso responde todas las consultas “sin herramientas”. Se mejora la query para el próximo ejemplo. Sin embargo, en la pregunta sobre las variedades de uvas agrega algunas que no están en el juego (al menos en la versión original).

TERCER EJEMPLO

```
    queries = [
        "¿Puedes darme información detallada de cada variedad de
uva del juego según el grafo del proyecto? Puntajes, tipos, etc",
        "Según el manual del juego, ¿cuáles son las acciones que se
pueden realizar en cada estación?",
        "Según el manual del juego, ¿puedes detallarme los
elementos que componen el juego? Cantidad de cartas, fichas,
tableros, etc.",
    ]
```

En este caso, responde las dos primeras preguntas sin usar las herramientas y luego responde la última con información en inglés, no precisamente relacionada con la consulta.

CUARTO EJEMPLO

```
    queries = [
        "¿Puedes darme información detallada de cada variedad de
uva 'Merlot'? ¿De qué tipo es? ¿Cuantos puntos suma?",
        "Puedes indicarme qué cartas de visitantes son de
'Verano'?",
        "¿Qué recibimiento tuvo el juego en el publico? Puedes
revisar reseñas en la base de datos de documentos.",
    ]
```

En estas consultas el agente comenzó a fallar por completo respondiendo en inglés, pero no con información del todo precisa sobre la pregunta realizada. En la segunda consulta indica que falla por time out.

CONCLUSIONES

No considero que haya obtenido un buen resultado en la implementación del agente, ya que no logro que utilice las herramientas provistas para la búsqueda en las bases de datos del proyecto. Las falencias de las funciones de la primera parte del trabajo, afectan el desempeño del agente.

A pesar de esto, para las preguntas que sí puede responder, es notable la mejora frente a los resultados obtenidos con el chatbot.

El uso de agentes permite integrar múltiples fuentes de información y herramientas, lo que debería optimizar las respuestas. Sin embargo, en esta implementación, aún es necesario mejorar la interacción con las bases de datos y ajustar el flujo de información para garantizar respuestas acertadas.

Aunque el agente representa una solución poderosa para la gestión del conocimiento, aún requiere ajustes importantes para aprovechar plenamente su potencial en este proyecto.