

UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y
AGRIMENSURA

TEC. INTELIGENCIA ARTIFICIAL

Procesamiento del Lenguaje Natural

Trabajo práctico N°2

PROFESOR A CARGO:
Dolores Sollberger

PROFESORES DE LA MATERIA:
Alan Geary
Juan Pablo Manson
Constantino Ferrucci

INTEGRANTES:
Alsop Agustín (A-4651/7)



2024

1. Índice

1. Índice.....	1
2. Introducción.....	3
2.1. Contexto.....	3
2.2. Objetivo.....	3
2.3. Contra tiempo.....	4
3. Metodología.....	5
3.1. Datos Utilizados.....	5
3.2. Herramientas y tecnologías.....	6
4. Desarrollo.....	7
4.1. Introducción.....	7
4.2. Resumen.....	7
4.3. Ejercicio 1.....	8
4.3.1. Introducción.....	8
4.3.2. Base de datos.....	9
4.3.2.1. Introducción.....	9
4.3.2.1. Generación de las base de datos.....	10
4.3.3. Clasificador.....	15
4.3.3.1. Introducción.....	15
4.3.3.2. Clasificador con Regresión Logística.....	16
4.3.3.3. Clasificador con LLM.....	18
4.3.3.4. Comparación.....	20
4.3.3.5. Database selector.....	20
4.3.4. Query Dinámica.....	21
4.3.4.1. Consultar Base de datos vectorial.....	21
4.3.4.2. Consultar Base de datos tabular.....	25
4.3.4.3. Consultar Base de datos de grafo.....	26
4.3.4.4. Consultor General.....	30
4.3.5. RAG (Retrieval-Augmented Generation).....	32
4.3.6. Menú Principal.....	35
4.4. Ejercicio 2.....	37
4.4.1. Introducción.....	37
4.4.2. Herramientas del Agente (Tool).....	38
4.4.3. Desarrollo.....	40
4.4.4. Resultados.....	40
4.4.5. Conclusiones.....	42
5. Conclusiones Generales.....	43
5.1. Conclusiones Generales.....	43
5.2. LLM utilizadas.....	43
5.3. Problemáticas Encontradas.....	46
5.4. Mejoras a Futuro.....	47

TRABAJO PRÁCTICO N° 2
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA
PROCESAMIENTO DEL LENGUAJE NATURAL
ROSARIO - SANTA FE

6. Anexo.....	48
----------------------	-----------

2. Introducción

2.1. Contexto

En el marco del Trabajo Práctico Final, se ha desarrollado un proyecto que aplica técnicas avanzadas de procesamiento del lenguaje natural (NLP) y herramientas de recuperación de información aumentada (Retrieval-Augmented Generation, RAG). Este trabajo aborda el desafío de crear un sistema inteligente capaz de interactuar de manera fluida y precisa en el contexto de un juego de mesa del género Eurogame, utilizando diversas fuentes de datos como documentos de texto, bases de datos tabulares y grafos.

El juego seleccionado para este trabajo es Viticulture, un Eurogame estratégico donde los jugadores asumen el rol de propietarios de viñedos en la Toscana. A través de la planificación de recursos y la toma de decisiones, deben gestionar sus viñas, producir vinos y comercializarlos para lograr el éxito en un mercado competitivo. Viticulture se distingue por su temática inmersiva, sus mecánicas de colocación de trabajadores y la posibilidad de combinar estrategias a largo plazo con acciones inmediatas.

2.2. Objetivo

El principal objetivo del proyecto es diseñar e implementar un chatbot experto en Viticulture, empleando la técnica RAG para recuperar información de manera eficiente y responder a consultas del usuario en lenguaje natural, ya sea en español o inglés. Además, se busca incorporar un agente inteligente que aproveche herramientas específicas para integrar de forma dinámica las distintas fuentes de información en las respuestas generadas.

Este informe incluye una descripción detallada de las técnicas utilizadas, los resultados obtenidos y las mejoras potenciales que podrían implementarse para optimizar el desempeño del sistema.

2.3. Contra tiempo

El desarrollo de este Trabajo Práctico Final estuvo marcado por diversas dificultades relacionadas con la gestión del tiempo, lo que representó un desafío significativo para llevar adelante las tareas propuestas. Debido a la proximidad de los plazos de entrega y la complejidad técnica del proyecto, se optó por priorizar el entendimiento de las distintas interacciones y comunicaciones con los Modelos de Lenguaje Extendido (LLM) y las herramientas involucradas, más que en alcanzar resultados perfectos o completamente optimizados.

Esta decisión fue tomada con el objetivo de consolidar un aprendizaje profundo sobre los principios y fundamentos detrás de los sistemas RAG y los agentes basados en ReAct. Si bien algunos resultados obtenidos pueden no ser ideales, el enfoque principal se centró en comprender el funcionamiento y la implementación práctica de estas tecnologías, con miras a aplicarlas con mayor eficacia en futuros proyectos.

3. Metodología

3.1. Datos Utilizados

Para la realización de este proyecto, se recopilaron datos provenientes, principalmente, de la página web BoardGameGeek (BGG), una de las plataformas más importantes a nivel mundial dedicada a los juegos de mesa. BGG es un sitio web que actúa como una base de datos centralizada y comunidad en línea para los entusiastas de los juegos de mesa. En esta plataforma, los usuarios pueden encontrar información detallada sobre juegos, incluyendo descripciones, reglas, estadísticas, reseñas, foros de discusión y recursos adicionales como archivos descargables y videos.

En el caso particular de Viticulture, los datos extraídos incluyeron:

- Reglas del juego y guías disponibles en formato escrito (sección Files).
- Estadísticas y análisis del juego (sección Stats), que proporcionaron datos tabulares como rankings y puntuaciones.
- Reseñas y opiniones de expertos o jugadores experimentados, obtenidas de la sección de blogs y videos.

Estos datos fueron procesados y estructurados para alimentar las diferentes fuentes requeridas por el sistema (textos, tablas y grafos). La calidad y diversidad de los datos obtenidos de BGG resultaron fundamentales para garantizar un buen desempeño del modelo al responder preguntas relacionadas con el juego.

3.2. Herramientas y tecnologías

Para la realización del trabajo, se utilizaron las siguientes herramientas:

- **Python:** Python es un lenguaje de programación de alto nivel que destaca por su sintaxis clara y su facilidad de uso, lo que permite un desarrollo rápido y eficiente. Se utilizó principalmente para la implementación de scripts y modelos, aprovechando su amplia biblioteca de paquetes para manipulación de datos y análisis, así como su versatilidad en entornos de programación científica.
- **Google Colab:** Google Colab es una plataforma en la nube que permite programar y ejecutar código en Python de manera interactiva. Ofrece una infraestructura accesible para realizar cálculos complejos sin depender de recursos locales y facilita la colaboración en tiempo real. Esta herramienta fue clave para el desarrollo y prueba de los scripts, permitiendo ejecutar el código y visualizar los resultados sin necesidad de una instalación local.
- **Google Docs:** Google Docs es una herramienta de procesamiento de textos en línea que facilita la creación y edición de documentos. En este proyecto, se utilizó para redactar y estructurar el informe, permitiendo el trabajo colaborativo y la edición en tiempo real, además de ofrecer opciones de formato y revisión de contenido.
- **ChatGPT:** ChatGPT es una herramienta de inteligencia artificial desarrollada por OpenAI, que utiliza procesamiento de lenguaje natural para generar y entender texto. En este proyecto, se empleó para la creación de la base de datos `test_clasificador` y `train_clasificador` que se utilizaron para entrenar un modelo de clasificación.

4. Desarrollo

4.1. Introducción

En esta sección se detallan los pasos realizados para la implementación de las soluciones a los ejercicios planteados en el trabajo práctico. El desarrollo se divide en dos ejercicios principales.

Ambos ejercicios buscan poner en práctica conceptos clave de la materia, como el uso de LLMs, la generación de embeddings, y la creación de flujos dinámicos para la consulta de datos en distintos formatos. El objetivo no solo es obtener un chatbot funcional, sino también profundizar en la comprensión de los sistemas RAG y ReAct, priorizando la integración de herramientas y técnicas aprendidas durante el curso.

4.2. Resumen

La aplicación diseñada permite al usuario interactuar con un sistema avanzado de preguntas y respuestas, estructurado en dos niveles de funcionalidad:

1. Chatbot experto sobre Viticulture (Ejercicio 1):

El usuario puede iniciar una conversación en español o inglés para consultar sobre cualquier aspecto del juego Viticulture, como reglas, mecánicas o información sobre su diseño.

- Flujo de interacción:

- El usuario formule una pregunta en lenguaje natural.
- El sistema clasifica la consulta para identificar la fuente de datos relevante (documentos, tablas o base de datos de grafos).
- Luego, el chatbot genera una respuesta basada en la información extraída, presentándola de manera clara y en el idioma de la consulta.

- Ejemplo:

Usuario: *¿Cómo funcionan las cartas de verano en Viticulture?*

Respuesta: *Las cartas de verano permiten realizar acciones específicas que optimizan la producción de vino, como plantar viñas o ganar bonificaciones adicionales. Puedo proporcionarte más detalles si lo necesitas.*

Nota: Este ejemplo es meramente ilustrativo, no asemeja a los resultados finales

2. Agente basado en ReAct (Ejercicio 2):

En este nivel, el sistema opera como un agente capaz de utilizar herramientas específicas para resolver preguntas más complejas o combinadas que requieren información de múltiples fuentes.

- Flujo de interacción:
 - El usuario plantea una consulta que involucra información de documentos, tablas y grafos.
 - El agente selecciona las herramientas necesarias (*doc_search*, *table_search* y/o *graph_search*), realiza búsquedas dinámicas y combina los resultados para generar una respuesta coherente y completa.

- Ejemplo:

Usuario: *¿Quién diseñó Viticulture y cuáles son los puntajes promedio obtenidos en las partidas según BGG?*

Respuesta: *El juego Viticulture fue diseñado por Jamey Stegmaier y Alan Stone. Según los datos extraídos de BGG, el puntaje promedio de las partidas es de 8.2 sobre 10.*

Nota: Este ejemplo es meramente ilustrativo, no asemeja a los resultados finales

4.3. Ejercicio 1

4.3.1. Introducción

En este ejercicio se desarrolla un sistema de generación de respuestas aumentadas por recuperación (Retrieval-Augmented Generation, RAG) diseñado para responder preguntas relacionadas con el juego Viticulture. El objetivo principal es construir una herramienta capaz de proporcionar información precisa y relevante utilizando múltiples fuentes de datos, tales como documentos, tablas y bases de datos de grafos.

El proceso incluye varias etapas: preparación de datos, creación de las distintas fuentes de conocimiento, clasificación de preguntas según el tipo de información requerida y generación de respuestas mediante modelos avanzados. Además, se implementan y evalúan dos enfoques diferentes para el clasificador de preguntas, analizando y comparando sus desempeños.

Este sistema busca no solo responder consultas específicas sobre el juego, sino también explorar la capacidad de los modelos RAG para integrar diversas fuentes de información en un entorno dinámico y eficiente.

4.3.2. Base de datos

4.3.2.1. Introducción

En esta sección se detalla la creación y estructura de las tres bases de datos necesarias para soportar el funcionamiento del sistema. Cada una está diseñada para abordar aspectos específicos de la información sobre el juego Viticulture, aprovechando las características particulares de cada tipo de base de datos:

1. **Base de datos vectorial:** Esta base está enfocada en almacenar información clave sobre el juego y su contexto, optimizada para búsquedas semánticas rápidas y precisas. Los datos se organizan en los siguientes tópicos:
 - a. Descripción general del juego.
 - b. Glosario de términos importantes.
 - c. Mecánicas principales.
 - d. Objetivos del juego.
 - e. Reglas generales.
 - f. Reglas específicas para el modo de un jugador.
 - g. Reseña del juego
2. **Base de datos de grafos:** Esta base de datos está diseñada para almacenar y analizar las relaciones entre las personas involucradas en la creación de Viticulture y sus expansiones. Se incluyen datos sobre diseñadores, ilustradores y otros colaboradores clave, destacando sus roles específicos dentro del desarrollo del juego. Además, la base refleja cómo estas personas están conectadas entre sí y cómo se vinculan con las expansiones del juego. Este enfoque permite modelar de manera intuitiva las conexiones jerárquicas y contextuales entre los diferentes actores y elementos relacionados con Viticulture, facilitando un análisis más dinámico y profundo de estas interacciones.
3. **Base de datos tabular:** Se compone de dos tablas principales:
 - a. Una tabla que contiene las estadísticas del juego base, como puntuaciones, tiempo estimado de juego, número de jugadores, entre otros datos relevantes.
 - b. Una tabla dedicada exclusivamente a las expansiones, con detalles como sus nombres, fechas de lanzamiento y características principales.

Esta división y organización permite que cada tipo de base de datos cumpla una función específica y complementaria, facilitando el acceso y la integración de los datos para las funcionalidades del sistema.

4.3.2.1. Generación de las base de datos

Base datos vectorial

Para la generación de la base de datos vectorial se utilizaron las reglas oficiales del juego, extraídas de la página oficial de *BoardGameGeek (BGG)*. Estas reglas fueron subdivididas en las secciones mencionadas previamente (descripción, glosario, mecánica, objetivo, reglas generales, reglas para un jugador y reseña), que se incorporaron como metadatos en la base vectorial. Esto permite realizar consultas de manera más eficiente y organizada.

Flujo de generación de datos

1. Carga de datos

Los documentos en formato PDF fueron descargados y procesados utilizando la librería de Python PyPDF2. A través de esta herramienta, se extrajeron los textos correspondientes a cada sección específica del reglamento, almacenados en variables independientes.

El [video review](#) fue procesado mediante la librería de python “youtube transcript api” mediante la misma se extrae en formato texto lo que se habla en el video.

2. Limpieza de texto

Para garantizar que los textos fueran adecuados para su uso, se aplicó un proceso de limpieza que elimina:

- Caracteres no codificados
- Caracteres no alfanuméricos
- Espacios múltiples
- Espacios iniciales y finales

3. Generación de chunk

El texto de cada sección fue dividido en fragmentos más pequeños con un tamaño de 500 caracteres cada uno, incluyendo un solapamiento de 50 caracteres entre fragmentos consecutivos. Esto permite mejorar la recuperación de información al mantener contexto entre los fragmentos.

4. Carga en la base de datos vectorial

La creación de la base vectorial se realizó utilizando la librería de Python chromadb. Cada fragmento de texto, generado en el paso anterior, fue cargado en la base junto con:

- Contenido textual: El texto del fragmento.
- Vector: La representación vectorial del fragmento.
- Metadatos: Información sobre la sección a la que pertenece el texto.

Vectorización

Para vectorizar los textos, se utilizó el modelo *Universal Sentence Encoder Multilingual* de la librería TensorFlow Hub. Este modelo fue elegido por su capacidad de codificación eficiente y su flexibilidad para trabajar con múltiples idiomas.

Base datos Tabular

Para la creación de la base de datos tabular se utilizó la sección de estadísticas (*stats*) del juego *Viticulture* y sus expansiones en la página de *BoardGameGeek* (BGG). El objetivo fue extraer información relevante para analizar las características del juego base y cada expansión. Los datos obtenidos son los siguientes:

1. **Puntaje promedio:** Calificación promedio asignada por los usuarios.
2. **Número en el rating:** Posición en el ranking global de juegos en BGG.
3. **Desviación estándar:** Medida de la variabilidad en las calificaciones del juego.
4. **Peso:** Nivel de complejidad del juego, evaluado por los usuarios.
5. **Número de comentarios:** Cantidad de reseñas o análisis escritos por los jugadores.
6. **Número de fans:** Usuarios que marcaron el juego como uno de sus favoritos.
7. **Número de visitas a la página:** Cantidad de veces que la página del juego ha sido consultada.
8. **Overall rank:** Posición general del juego en todas las categorías.
9. **Strategy rank:** Posición específica en la categoría de juegos de estrategia.
10. **All-time plays:** Número total de veces que el juego ha sido jugado.
11. **This month:** Número de veces que el juego ha sido jugado en el mes actual.
12. **Own:** Número de usuarios que tienen el juego en su colección.
13. **Prev owned:** Número de usuarios que anteriormente poseyeron el juego.
14. **For trade:** Usuarios que han puesto el juego en oferta de intercambio.
15. **Want in trade:** Usuarios interesados en adquirir el juego mediante intercambio.
16. **Wishlist:** Número de usuarios que han añadido el juego a su lista de deseos.
17. **Has parts:** Usuarios que tienen componentes o partes relacionadas con el juego.
18. **Want parts:** Usuarios interesados en obtener componentes o partes del juego.

Flujo de generación de datos

1. Extracción del texto

Para obtener estos datos, se realizó un *web scraping* utilizando la librería de Python selenium.

- Para el juego principal, las estadísticas fueron extraídas de la sección de estadísticas (*stats*).
- Para las expansiones, la información fue tomada de la sección correspondiente a expansiones.

Nota: Debido a que el nombre de las expansiones contenía el año de lanzamiento, se aplicó un proceso de separación para dividir el nombre y el año en dos columnas distintas.

2. Generación de tabla

Los datos extraídos a través del *web scraping* fueron organizados en una lista de diccionarios, donde cada entrada representaba las estadísticas de un juego. Posteriormente, utilizando la librería pandas, se transformaron en un *DataFrame*, creando así las tablas necesarias para realizar consultas.

- Una tabla contiene las estadísticas del juego base.
- Otra tabla recopila las estadísticas de todas las expansiones.

Justificación de no guardar los datos vectorizados como se hizo con *cromadb*

Los datos tabulares no se guardaron como vectores porque el tipo de información extraída está estructurada y es de naturaleza categórica o numérica, lo que no requiere representación en un espacio vectorial. Además, las consultas a realizar sobre esta información están orientadas a la búsqueda directa y análisis estadístico, tareas que se manejan de forma más eficiente con tablas estructuradas. La vectorización es útil para trabajar con textos o datos no estructurados, pero no aporta ventajas significativas en este caso.

Base datos Grafo

Para la creación de la base de datos de grafo se utilizó la sección de créditos en la página de *BoardGameGeek (BGG)*. En esta sección, se recopilaron los datos de las personas que participaron en el desarrollo del juego principal y sus expansiones, junto con los roles específicos que desempeñan.

Roles registrados

A continuación, se detallan los roles posibles, agrupando las variantes en singular y plural para evitar repeticiones:

- **Designers:** Diseñadores responsables de la creación del juego y sus mecánicas.
- **Solo Designers:** Diseñadores individuales que trabajaron en solitario en el desarrollo del juego.
- **Artists:** Ilustradores encargados del arte visual del juego, como portadas, tableros, y cartas.
- **Developers:** Desarrolladores que trabajaron en las mecánicas y reglas finales del juego.
- **Graphic Designers:** Diseñadores gráficos que trabajaron en el diseño visual y la experiencia de usuario.
- **Sculptors:** Escultores responsables de las miniaturas u otros elementos tridimensionales del juego.
- **Editors:** Editores encargados de revisar el contenido escrito del juego, como reglas e instrucciones.
- **Writers:** Escritores que trabajaron en textos narrativos, descripciones, o trasfondo del juego.
- **Insert Designers:** Diseñadores que crearon elementos organizativos, como inserts para las cajas del juego.

Nota: Los valores plurales y singulares han sido agrupados para garantizar que no haya duplicación al momento de construir el grafo.

Flujo de generación de datos

1. Extracción del texto

El proceso de extracción de datos se realizó de manera similar al de la base de datos tabular:

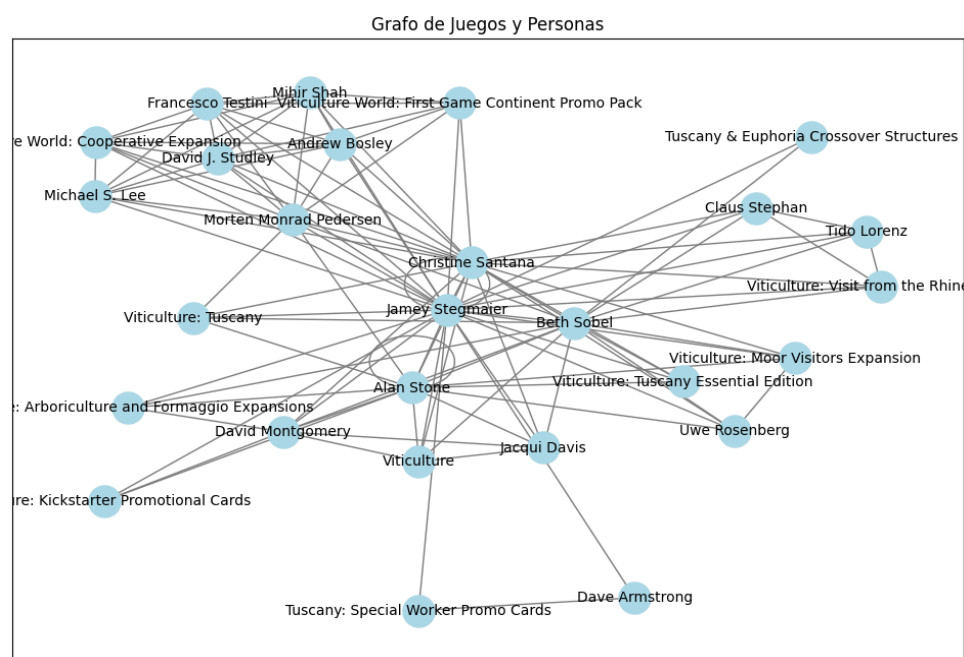
- Se iteró a través de las URLs de cada juego en la página de BGG.
- Para cada juego, se generó un diccionario que incluía los roles y las personas que desempeñaron dichos roles. Este diccionario es la base para construir las conexiones.

2. Generación del grafo

Una vez obtenido el diccionario, se construyó el grafo siguiendo estos pasos:

1. Cada juego y cada persona se convirtieron en nodos del grafo.
2. Las conexiones (aristas) se generaron con base en las relaciones entre los juegos y las personas. Por ejemplo:
 - Una persona que participó en el desarrollo de varios juegos estará conectada con cada uno de ellos.
 - Los juegos en los que trabajó un mismo equipo de personas estarán conectados de manera indirecta a través de dichas personas.
3. Durante la creación de los nodos, se verificó si una persona ya había sido añadida al grafo para evitar duplicaciones y garantizar la integridad del modelo.

Resultado del Grafo generado



Justificación de no guardar los datos vectorizados como se hizo con cromadb

Los datos de la base de datos de grafo no se guardaron como vectores porque su estructura se centra en nodos y conexiones, no en información textual que requiera representación en un espacio vectorial.

Razones específicas:

1. **Relaciones contextuales:** El objetivo del grafo es analizar y representar las relaciones jerárquicas y colaborativas entre personas y juegos. Los vectores no son eficientes para este tipo de análisis relacional.
2. **Eficiencia en consultas:** Los grafos permiten realizar consultas como "¿Quiénes trabajaron en el diseño de ambos juegos?" o "¿Qué personas conectan el juego base con una expansión?" de manera más directa y eficiente que una representación vectorial.

En conclusión, la estructura de un grafo es la representación más adecuada para capturar las relaciones entre personas, roles y juegos, ofreciendo flexibilidad y precisión para las consultas relacionales.

4.3.3. Clasificador

4.3.3.1. Introducción

En esta sección se describe el proceso de clasificación de una consulta del usuario (query) en función de su contenido y cómo se dirige posteriormente a la base de datos correspondiente para obtener la información deseada. Este sistema clasifica la consulta en uno de los siguientes tópicos:

- Descripción
- Glosario
- Mecánica
- Objetivo
- Reglas generales
- Reglas para un jugador
- Stats del juego
- Stats de las expansiones
- Personas involucradas en el desarrollo del juego
- Reseñas

Una vez clasificado el tópico, un componente llamado *"data_base_class"* determina a qué base de datos (vectorial, tabular o de grafos) debe dirigirse para obtener la información.

4.3.3.2. Clasificador con Regresión Logística

El primer enfoque para clasificar las consultas utiliza un modelo de regresión logística entrenado con datos etiquetados manualmente. Este modelo se diseñó para asociar palabras clave y patrones textuales comunes en las consultas de los usuarios con los tópicos mencionados anteriormente.

Proceso de entrenamiento y funcionamiento:

1. Preparación de datos:

- Se recopiló un conjunto de consultas simuladas.
- Cada consulta fue etiquetada con el tópico correspondiente.
- Se obtuvieron 50 preguntas por tópico para entrenamiento y 10 preguntas por tópico para test

2. Vectorización del texto:

- Las consultas fueron vectorizadas utilizando la misma técnica que con la base de datos vectorial

3. Entrenamiento:

- Se entrenó el modelo de regresión logística con las consultas vectorizadas como entradas y los tópicos como etiquetas.

4. Clasificación:

- Cuando el usuario ingresa una consulta, el modelo predice el tópico con base en la probabilidad calculada para cada categoría.

Ventajas:

- Ligero y eficiente en términos computacionales.
- Adecuado para sistemas donde el procesamiento rápido es clave.

Desventajas:

- Limitado a patrones predefinidos en los datos de entrenamiento.
- Menos efectivo en consultas complejas o ambiguas.

TRABAJO PRÁCTICO N° 2
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA
PROCESAMIENTO DEL LENGUAJE NATURAL
ROSARIO - SANTA FE

Resultados

	precision	recall	f1-score	support
componentes	0.82	0.90	0.86	10
descripcion	0.67	0.80	0.73	10
glosario	1.00	0.90	0.95	10
mecanica	0.62	0.50	0.56	10
objetivo	0.80	0.80	0.80	10
reglas generales	0.75	0.90	0.82	10
reglas un jugador	0.88	0.70	0.78	10
reseña	1.00	1.00	1.00	10
accuracy			0.81	80
macro avg	0.82	0.81	0.81	80
weighted avg	0.82	0.81	0.81	80

Los resultados obtenidos mediante la regresión lineal fueron muy positivos, alcanzando un accuracy del 81%. Algunas clases lograron clasificar de manera casi perfecta en el grupo de test. Sin embargo, se observaron variaciones en el desempeño entre las diferentes clases. La clase con menor rendimiento fue **Mecánica**, con un recall del 50%, mientras que la clase con mejor desempeño fue **Reseña**, logrando un recall perfecto del 100%. Estos resultados reflejan un buen ajuste del modelo, aunque con margen de mejora en clases específicas.

```
texto_prueba = ['como juego la parte del invierno']
texto_prueba = embed(texto_prueba)
clasificador_consulta.predict(texto_prueba)

array(['descripcion'], dtype=object)
```

Se detectó un problema al evaluar el modelo con consultas más complejas: a medida que aumenta la dificultad de las preguntas, el modelo tiende a fallar con mayor frecuencia. Esto indica que su capacidad para manejar consultas complejas es limitada, lo que reduce su flexibilidad y robustez en escenarios más exigentes.

4.3.3.3. Clasificador con LLM

El segundo enfoque implementado para la clasificación de tópicos utiliza un modelo de lenguaje grande (LLM), específicamente **Qwen**. Este modelo destaca por su capacidad de comprensión avanzada del lenguaje natural y su habilidad para procesar y contextualizar consultas complejas.

¿Qué es Qwen?

Qwen es un modelo LLM de alto rendimiento desarrollado para tareas de procesamiento de lenguaje natural (NLP). Diseñado para manejar consultas con un entendimiento profundo del contexto y los matices del lenguaje, Qwen está preentrenado en un extenso corpus de datos, lo que le permite generar respuestas precisas y relevantes. Es ampliamente utilizado en aplicaciones como análisis de texto, generación de lenguaje natural y clasificación de información.

Funcionamiento:

1. Entrada de la consulta:

- El usuario ingresa una consulta en lenguaje natural, que puede ser compleja o ambigua.

2. Procesamiento por Qwen:

- La consulta es enviada al modelo Qwen, que evalúa el contenido y contexto del texto. Basándose en su conocimiento previo, el modelo identifica el tópico al que pertenece la consulta.

3. Integración con el Database Selector:

- Una vez identificado el tópico, Qwen pasa esta información al *Database Selector*, que determina qué base de datos (vectorial, tabular o de grafos) debe ser consultada para responder a la solicitud del usuario.

Ventajas del Clasificador basado en Qwen

- Mayor precisión en consultas complejas o ambiguas: Gracias a su entrenamiento avanzado, Qwen puede interpretar correctamente consultas que no siguen una estructura rígida o que contienen matices del lenguaje.
- No requiere entrenamiento adicional: Qwen está preentrenado en un vasto corpus de datos, lo que elimina la necesidad de entrenar modelos personalizados.
- Flexibilidad y adaptabilidad: Puede manejar un rango diverso de tareas relacionadas con el lenguaje natural en distintos idiomas.

Desventajas del Clasificador basado en Qwen

- Consumo de recursos computacionales más alto: Al ser un modelo avanzado, su implementación requiere mayor capacidad de cómputo en comparación con los clasificadores tradicionales.
- Dependencia de conectividad: Si Qwen no está alojado localmente, su funcionamiento depende de una conexión estable con el servidor donde se encuentra desplegado.
- La velocidad de respuesta no es rápida

El uso de Qwen como clasificador permite aprovechar las últimas innovaciones en NLP, ofreciendo una solución robusta y precisa para la clasificación de tópicos en sistemas complejos. Sin embargo, su implementación debe ser considerada en función de los recursos disponibles y las necesidades específicas del sistema.

```
llm_class_client = InferenceClient(api_key="hf_pnbJlCjBpTyJDFAxepMNUVtBVvhMfppTZ")  
query_str = "como juego la parte del invierno"  
print(data_base_class(llm_class_topic(query_str, llm_class_client)))  
mecanica
```

En este caso, en comparación con la regresión logística, el modelo LLM logró interpretar correctamente qué se trataba de una mecánica del juego, demostrando una mayor flexibilidad y una mejor comprensión del lenguaje natural. Esto resalta su capacidad para captar la intención detrás de la consulta, incluso cuando se utilizan expresiones menos estructuradas o ambiguas.

4.3.3.4. Comparación

Comparación entre los dos clasificadores

Aspecto	Regresión Logística	LLM
Velocidad	Alta	Media
Precisión en consultas complejas	Baja	Alta
Requerimientos computacionales	Bajos	Altos
Flexibilidad	Limitada	Amplia

Ambos clasificadores tienen aplicaciones específicas según el contexto. El clasificador basado en regresión logística es ideal para sistemas rápidos con consultas estructuradas, mientras que el clasificador basado en LLM se adapta mejor a consultas en lenguaje natural o escenarios más complejos.

Para la realización de este trabajo se utilizó el modelo de LLM para clasificar.

4.3.3.5. Database selector

Una vez que se clasifica el tópico, el *Database Selector* (`data_base_class`) dirige la consulta a la base de datos correspondiente como se detalló anteriormente:

- **Base vectorial:** Para tópicos relacionados con las reglas, glosarios, mecánicas u objetivos.
- **Base tabular:** Para consultas sobre estadísticas del juego principal y expansiones.
- **Base de grafos:** Para tópicos relacionados con las personas involucradas en el desarrollo del juego.

Este flujo asegura una interacción eficiente y precisa con el sistema de bases de datos, optimizando tanto el rendimiento como la experiencia del usuario.

4.3.4. Query Dinámica

En esta sección se explicará cómo se desarrolla el proceso para que, una vez seleccionada la base de datos y el tópico correspondiente, nuestro programa realice una consulta a la base de datos designada.

El flujo se divide en tres tipos de consulta, según el tipo de base de datos:

1. Base de datos vectorial.
2. Base de datos tabular.
3. Base de datos de grafo.

Además, se incluye una función general llamada "Questioner", que combina las consultas de los diferentes tipos de bases de datos, integrando todas estas funciones en un único sistema dinámico y robusto.

4.3.4.1. Consultar Base de datos vectorial

Consulta a la Base de Datos Vectorial

Una vez clasificado el tópico y seleccionada la base de datos vectorial, se utiliza ChromaDB como motor para realizar la consulta. Este proceso se realiza con base en los siguientes parámetros:

1. Query vectorizada:

- La consulta es transformada en un vector utilizando la misma técnica de vectorización que fue aplicada durante la construcción de la base de datos vectorial.

2. Número de resultados:

- Especifica cuántas respuestas queremos que devuelva la base de datos. Por ejemplo, se puede solicitar el top 5 de documentos más relevantes.

3. Filtro de metadata:

- Se pasa un diccionario que contiene los valores de la metadata, permitiendo filtrar los resultados para que correspondan específicamente al tópico seleccionado por el clasificador.

Al realizar la consulta, ChromaDB devuelve un diccionario con las siguientes claves:

- Documents: Los textos asociados que coinciden con la consulta.
- IDs: Identificadores únicos de los documentos.
- Metadata: Información contextual asociada a cada documento, como el tópico, categoría o etiquetas específicas.
- Distances: Las distancias calculadas entre el vector de la consulta y los vectores de los documentos almacenados, indicando el grado de similitud.

Reranker

¿Por qué utilizar un Reranker?

El reranker es una herramienta fundamental para refinar los resultados de una base de datos vectorial. Si bien ChromaDB ya devuelve resultados ordenados por relevancia, el reranker permite:

- Reevaluar la similitud de los textos: Utilizando técnicas avanzadas de procesamiento de lenguaje natural (NLP).
- Asignar un puntaje de relevancia más preciso: Basándose en el contexto semántico de los documentos.
- Optimizar los resultados para el usuario final: Asegurándose de que los documentos más relevantes estén al principio de la lista.

Flujo con el Reranker

1. Recepción de resultados iniciales:

- De los resultados devueltos por ChromaDB, se toma únicamente la sección "Documents", que contiene los textos relacionados con la consulta.

2. Asignación de puntajes:

- Cada texto pasa por el reranker, que le asigna un puntaje de similitud en función de la consulta original.

3. Selección de los mejores documentos:

- Se seleccionan los tres documentos con mejor puntaje.

4. Combinación de textos:

- Los textos seleccionados se combinan en un único cuerpo de texto cohesivo. Esto asegura que toda la información relevante esté consolidada para la siguiente etapa de procesamiento.

5. Salida final:

- El texto combinado se devuelve como salida, listo para ser procesado por el modelo LLM que responderá al usuario.

El uso de esta metodología no solo garantiza precisión en las respuestas, sino que también optimiza el flujo de información, logrando un balance entre eficiencia (ya que no se extrae todo el dataset) y relevancia (ya que solo se traen los más relevantes de los mismo) en los resultados entregados al usuario.

Ventajas del Enfoque

1. Precisión en la recuperación de información:

- Al utilizar una query vectorizada junto con un filtro de metadata, el sistema puede encontrar documentos altamente relevantes al contexto de la consulta, reduciendo resultados irrelevantes.

2. Personalización mediante filtros:

- Los filtros de metadata permiten especificar criterios adicionales para refinar aún más las búsquedas, garantizando que los resultados sean específicos a las necesidades del usuario.

3. Optimización mediante reranking:

- El uso de un reranker asegura que las respuestas no solo sean relevantes, sino que estén ordenadas por un puntaje de similitud, mejorando la calidad del resultado final.

4. Flexibilidad en las consultas:

- Al vectorizar las queries utilizando técnicas consistentes, el sistema es capaz de adaptarse a diferentes tipos de consultas, ya sean preguntas directas o búsquedas más abstractas.

5. Escalabilidad:

- Este enfoque puede escalar a grandes volúmenes de datos sin perder eficiencia, ya que las bases vectoriales están diseñadas para manejar grandes cantidades de documentos y realizar búsquedas rápidas.

Aplicación Práctica (Extraída del código)

Imaginemos que un usuario quiere obtener información sobre cómo se juega:
“¿Cómo se juega?”

1. Vectorización de la Query:

- La consulta en lenguaje natural es vectorizada utilizando la misma técnica empleada para indexar los documentos en la base de datos.

2. Consulta a la Base de Datos Vectorial:

- La query vectorizada se envía a la base de datos, junto con un filtro de metadata que limita los resultados a documentos clasificados en la categoría "reglas generales".
- Se especifica que se desean los 5 documentos más relevantes.

3. Reranking:

- Los documentos devueltos se procesan con el reranker, que asigna puntajes de similitud a cada documento en función de la consulta original.
- Los 3 mejores documentos se seleccionan y combinan en un solo texto.

4. Respuesta Final (ejemplo real):

- El sistema genera un texto consolidado:
 - “Se juega colocando a un trabajador en las casillas de acción para jugar cartas de visitante de verano o invierno. No se necesitan trabajadores adicionales para llevar a cabo la acción descrita en la carta. Solo está permitido jugar una carta de visitante si eres capaz de completar el efecto de la carta.”

Conclusión

El enfoque basado en bases vectoriales proporciona una herramienta poderosa para realizar búsquedas semánticas precisas y rápidas, incluso en grandes conjuntos de datos. La combinación de consultas vectorizadas, filtros de metadata y reranking no solo mejora la calidad de las respuestas, sino que también permite adaptarse a diferentes dominios de aplicación.

Este sistema es ideal para entornos donde la rapidez y la relevancia de la información son críticas, como investigación científica, documentación técnica o consultas de usuarios finales en plataformas interactivas. Su flexibilidad y escalabilidad aseguran que puede seguir siendo efectivo a medida que crece el volumen de datos o cambian las necesidades de los usuarios.

4.3.4.2. Consultar Base de datos tabular

En esta aplicación, contamos con dos bases de datos tabulares, tal como se mencionó previamente. Ambas utilizan el mismo flujo de consulta, diseñado para garantizar eficiencia y precisión en la obtención de información.

El proceso comienza cuando el Data Selector determina cuál de las dos bases de datos debe ser consultada en función del tópico identificado previamente. Una vez seleccionada la base de datos, la consulta se realiza de la siguiente manera:

Flujo de Consulta a la Base de Datos Tabular

1. Interpretación de la Consulta mediante LLM (Qwen):

- Se utiliza un modelo de lenguaje grande (LLM), en este caso Qwen, para interpretar la consulta del usuario.
- Qwen recibe como entrada el texto de la consulta junto con la estructura de la tabla (filas y columnas). Su tarea es identificar a qué columna y fila de la tabla hace referencia la pregunta del usuario.

2. Búsqueda en el DataFrame:

- Una vez que Qwen decide qué celda de la tabla contiene la información relevante, se realiza una búsqueda en el DataFrame que representa la base de datos tabular.

3. Construcción de la Respuesta:

- El sistema devuelve un texto que contiene:
 - El valor exacto encontrado en la tabla.
 - La columna de donde se extrajo el dato, para que el usuario tenga un contexto claro sobre la información obtenida.

Ventajas del Enfoque

● Eficiencia en la consulta:

- Este proceso elimina la necesidad de que el usuario conozca la estructura exacta de las tablas, ya que el modelo LLM interpreta la consulta y selecciona los datos relevantes automáticamente.

● Precisión en la búsqueda:

- Al combinar la capacidad de interpretación contextual del LLM con la exactitud del DataFrame, se garantiza que la información recuperada sea precisa y contextualizada.

● Flexibilidad:

- Este sistema permite manejar consultas complejas, incluso cuando el usuario no formula su pregunta de manera directa o específica.

Aplicación Práctica (Extraída del código)

Por ejemplo, si un usuario pregunta:

“¿Qué puntaje tiene el juego viticulture?”

1. El LLM analizará la consulta y determinará qué debe buscar en una tabla que contiene información sobre las estadísticas del juego principal.
2. Seleccionará la columna correspondiente (por ejemplo, “Puntaje Promedio”) y buscará la columna en la tabla seleccionada.
3. Devolverá una respuesta del tipo:
 - *“Avg. Rating, 7.486”*

Conclusión

Este enfoque combina la potencia de los modelos LLM con estructuras tabulares tradicionales para proporcionar un sistema robusto y flexible. De esta manera, las bases de datos tabulares se pueden consultar de manera eficiente, con respuestas claras y precisas para cualquier tipo de consulta que el usuario realice.

4.3.4.3. Consultar Base de datos de grafo

En esta sección detallaremos el proceso para realizar consultas a la base de datos de grafos. Este tipo de consultas son las más complejas de las tres, debido a la naturaleza altamente interconectada de los datos y a la diversidad de preguntas que los usuarios pueden realizar.

El enfoque está diseñado para interpretar y procesar preguntas estructuradas en torno a las relaciones entre personas, roles, juegos y expansiones. Antes de implementar las consultas, se definieron las clases principales de preguntas que se pueden hacer al grafo:

Clases de Consultas al Grafo

- 1. rol_persona:**
 - Pregunta sobre qué rol desempeñó una persona en un juego específico.
 - Ejemplo: *“¿Qué rol tuvo Alan en el juego Viticulture?”*
- 2. personas_por_rol:**
 - Pregunta sobre quiénes desempeñan un rol específico en un juego o expansión.
 - Ejemplo: *“¿Quiénes fueron los diseñadores del juego Viticulture?”*
- 3. personas_por_juego:**
 - Pregunta sobre todas las personas que trabajaron, desarrollaron o estuvieron involucradas en un juego en particular.
 - Ejemplo: *“¿Quiénes trabajaron en el juego Viticulture?”*

4. juegos_por_persona:

- Pregunta sobre los juegos en los que participó una persona.
- Ejemplo: “¿En qué juegos trabajó Jane Viticulture?”

5. personas_por_rol_en_juego:

- Pregunta sobre las personas que desempeñaron un rol específico en un juego.
- Ejemplo: “¿Quién fue el productor del juego Viticulture?”

6. personas_que_trabajaron_con:

- Pregunta sobre las personas con las que trabajó una persona en cualquier juego.
- Ejemplo: “¿Con quién trabajó John Smith en todos los juegos?”

Proceso de Consulta al Grafo

1. Interpretación de la Consulta con LLM (Qwen):

- Cuando el usuario ingresa una consulta en lenguaje natural, se utiliza un modelo LLM, en este caso **Qwen**, para identificar a cuál de las clases de consulta mencionadas corresponde la pregunta.
- El modelo analiza las palabras clave y el contexto para clasificar correctamente la consulta.

2. Extracción de Nombres (Juego y Persona):

- Si la consulta incluye referencias a un **juego**, una **persona**, o ambas, pasa por una función adicional que utiliza el mismo LLM para extraer los nombres específicos mencionados en la consulta.
- Ejemplo: Si el usuario pregunta: “¿Quién fue el diseñador de Viticulture?”, el LLM identificará que "Viticulture" es un juego y "diseñador" es el rol.
- Al LLM se le ingresa dos listas con las opciones que puede elegir de juego y de roles para mantener la integridad del código y focalizar la respuesta en un dato que puede ser consultado al grafo

3. Filtrado y Procesamiento con networkx:

- Una vez clasificada la consulta y extraídos los nombres relevantes, el sistema utiliza funciones específicas desarrolladas con la librería **NetworkX (nx graph)** de Python para consultar el grafo.
- Dependiendo del tipo de pregunta, se aplican filtros al grafo para obtener la información requerida:
 - **Nodos:** Personas, juegos, roles.
 - **Relaciones:** Participación en juegos, colaboraciones, roles desempeñados.

4. Respuesta en Formato Texto:

- El resultado del filtrado se devuelve en formato texto, estructurado de manera clara y comprensible para el usuario.

Ventajas del Enfoque

- **Flexibilidad:**
 - Permite realizar consultas complejas que combinan múltiples elementos, como juegos, personas y roles.
- **Interfaz Natural:**
 - Los usuarios no necesitan conocer la estructura interna del grafo ni los nombres exactos de nodos o relaciones, ya que el LLM interpreta la intención de la consulta.
- **Escalabilidad:**
 - El sistema puede adaptarse a grafos más grandes o más densos sin necesidad de redefinir el proceso de consulta.

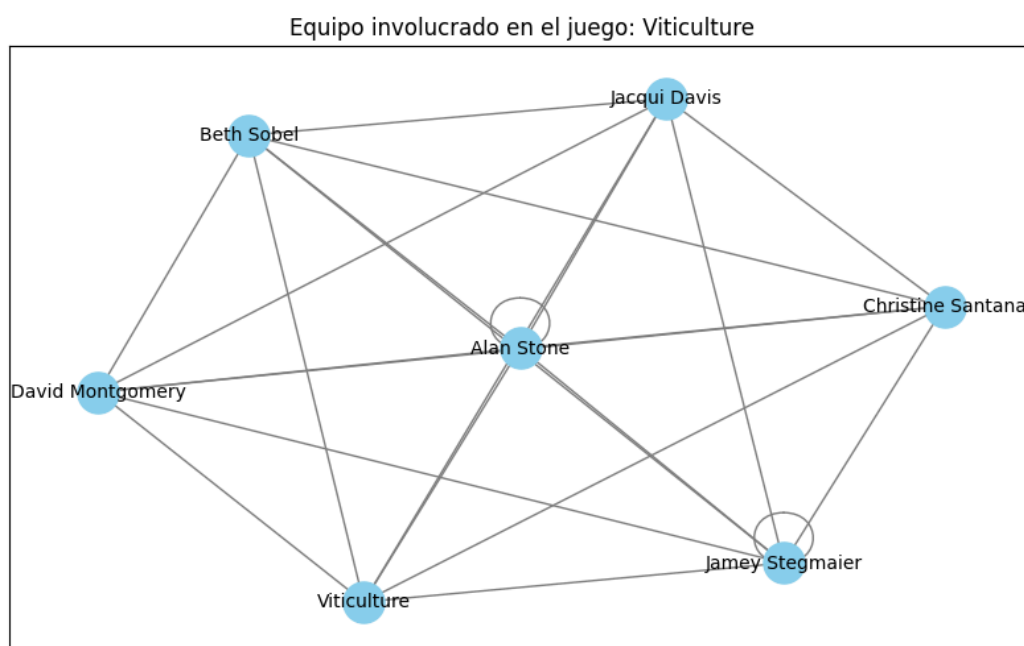
Aplicación Práctica (Extraída del código)

Supongamos que un usuario pregunta:

“¿Quienes trabajaron en viticulture?”

1. El LLM clasifica la consulta como **personas_por_juego** y detecta que el juego es “Viticulture”.
2. Se filtra el grafo utilizando estos criterios, buscando personas conectadas al nodo del juego “Viticulture” .
3. Se devuelve una respuesta en formato texto:
 - *“Alan Stone, Jamey Stegmaier, Beth Sobel, David Montgomery, Jacqui Davis, Christine Santana”*

Ejemplo gráfico del filtro:



Conclusión

El sistema de consultas al grafo combina la capacidad de interpretación avanzada de los LLM con la eficiencia de la librería NetworkX, proporcionando una herramienta poderosa para extraer información compleja y profundamente interrelacionada. Este enfoque no solo facilita la interacción con los datos, sino que también abre la posibilidad de realizar análisis más detallados y personalizados para los usuarios.

4.3.4.4. Consultor General

El **Consultor General (Questioner)** actúa como el componente central que integra y coordina todo el proceso de consulta explicado en las secciones anteriores. Este módulo se encarga de gestionar las interacciones entre el clasificador, el selector de base de datos y las funciones específicas de consulta, para garantizar una respuesta eficiente y precisa basada en la entrada del usuario.

Funcionamiento

1. **Recepción de la consulta del usuario:**
 - El usuario ingresa una consulta en lenguaje natural que el sistema debe procesar para devolver la información solicitada.
2. **Clasificación de la consulta:**
 - La consulta pasa primero por el **Clasificador LLM**, descrito en el apartado 4.3.3.3., que identifica el tópico al que pertenece la consulta, como *"descripción"*, *"reglas generales"*, o *"personas involucradas en el desarrollo del juego"*.
3. **Selección de la base de datos:**
 - Con el tópico identificado, el resultado es enviado al **Database Selector**, detallado en el apartado 4.3.3.5., que determina cuál de las tres bases de datos (vectorial, tabular o de grafo) es la más adecuada para responder a la consulta.
4. **Ejecución de la consulta:**
 - Una vez seleccionada la base de datos y con el tópico identificado, el sistema activa la función específica para realizar la consulta:
 - Si la base seleccionada es **vectorial**, se ejecuta el procedimiento descrito en el apartado 4.3.4.1.
 - Si la base seleccionada es **tabular**, se sigue el proceso detallado en el apartado 4.3.4.2.
 - Si la base seleccionada es **de grafo**, se utiliza el enfoque explicado en el apartado 4.3.4.3.
 - A la función correspondiente se le pasa la query vectorizada o procesada, junto con los parámetros necesarios (como filtros de metadata o columnas específicas).

5. **Devolución de la respuesta:**

- La función seleccionada retorna la información relevante, ya sea en forma de un texto consolidado, un valor específico, o una lista estructurada de resultados.
- El sistema genera una respuesta final que es enviada al usuario, asegurándose de que sea comprensible y directamente aplicable.

Ventajas del Consultor General

1. **Automatización Completa:**

- El usuario interactúa únicamente con una interfaz sencilla, mientras que el Consultor General gestiona todos los pasos necesarios de manera transparente.

2. **Versatilidad:**

- Gracias a su capacidad para integrar diferentes tipos de bases de datos, el sistema puede responder a una amplia variedad de consultas, desde búsquedas semánticas complejas hasta respuestas directas basadas en datos tabulares o relaciones de grafo.

3. **Escalabilidad Modular:**

- Al estar compuesto por módulos independientes, es posible añadir nuevas bases de datos o ajustar los clasificadores y funciones de consulta sin afectar al resto del sistema.

4. **Respuesta Precisa y Contextualizada:**

- La combinación del clasificador, selector de bases de datos y funciones específicas asegura que las respuestas sean relevantes al contexto y alineadas con la intención del usuario.

Conclusión

El **Consultor General** es la pieza clave que permite la interacción fluida entre el usuario y el sistema de información, consolidando todos los componentes desarrollados en este proyecto. Su diseño modular e integrado asegura que cualquier consulta pueda ser procesada de manera eficiente y precisa, independientemente de su complejidad o del tipo de datos requeridos.

Esta unificación no solo mejora la experiencia del usuario, sino que también proporciona una solución robusta y escalable para entornos donde el acceso rápido y relevante a la información es crítico.

4.3.5. RAG (Retrieval-Augmented Generation)

Introducción al RAG

El modelo **RAG (Retrieval-Augmented Generation)** es una técnica híbrida que combina la capacidad de recuperación de información de bases de datos con la generación de texto de modelos de lenguaje. En este caso, utilizamos el modelo **Zephyr** mediante una **API REST**, lo que permite una integración sencilla y eficiente con nuestro sistema.

El objetivo principal del RAG en este sistema es generar respuestas precisas y contextualizadas a las consultas de los usuarios, basándose no solo en la pregunta realizada, sino también en un contexto relevante obtenido previamente del sistema.

Ventajas de utilizar la API de Zephyr

1. **Facilidad de Implementación:**
 - La API REST de Zephyr permite una conexión rápida y directa con el modelo, sin necesidad de gestionar infraestructuras adicionales o entrenar modelos personalizados.
2. **Actualizaciones Continuas:**
 - Al utilizar un servicio externo, el modelo Zephyr se mantiene actualizado y mejora con el tiempo, asegurando respuestas más precisas y adaptadas a las necesidades del usuario.
3. **Escalabilidad y Rendimiento:**
 - La arquitectura de API permite manejar múltiples solicitudes concurrentes sin afectar el rendimiento del sistema.
4. **Compatibilidad:**
 - La API de Zephyr puede integrarse fácilmente con el flujo del sistema, recibiendo tanto el contexto como la query en un formato estándar, devolviendo respuestas que pueden ser procesadas sin complicaciones adicionales.
5. **Reducción de Costos Computacionales:**
 - Al delegar el procesamiento al servicio Zephyr, se evita el uso intensivo de recursos locales, optimizando así la carga del sistema.

Función del RAG en este sistema

El RAG actúa como el componente final encargado de generar una respuesta detallada y coherente. Una vez que el sistema ha clasificado la consulta, seleccionado la base de datos adecuada y recuperado el contexto más relevante, el RAG utiliza esta información para:

- **Entender la intención del usuario:** Interpretar la pregunta en lenguaje natural para responder de manera precisa.
- **Incorporar el contexto recuperado:** Garantizar que la respuesta sea específica y directamente aplicable al escenario del usuario.
- **Generar texto de manera natural:** Proporcionar respuestas completas que sean fáciles de entender y procesar por el usuario.

Flujo del Sistema con RAG

1. **Ingreso de la consulta:**
 - El usuario realiza una pregunta en lenguaje natural.
2. **Clasificación de la consulta:**
 - El sistema pasa la consulta por el **Clasificador LLM**, que identifica el tópico y la base de datos relevante para la consulta.
3. **Recuperación del contexto:**
 - Dependiendo de la base de datos seleccionada (vectorial, tabular o de grafo), se extrae el contexto relevante mediante el proceso detallado en la sección **4.3.4. Query Dinámica**.
4. **Envío al RAG:**
 - Una vez obtenido el contexto, el sistema combina este contexto con la query original del usuario y envía ambos parámetros a la API REST de Zephyr.
5. **Generación de la respuesta:**
 - El modelo Zephyr procesa el contexto y la query para generar una respuesta completa y precisa, la cual es retornada al usuario.

Ventajas del Flujo con RAG

1. **Precisión:**
 - Al combinar el contexto con la query, el sistema asegura que la respuesta generada esté alineada con la intención del usuario y la información relevante del sistema.
2. **Personalización:**
 - La respuesta generada por el modelo Zephyr está ajustada a las necesidades específicas de cada consulta, logrando una experiencia de usuario más satisfactoria.

3. **Eficiencia:**

- Al integrar el RAG como parte del flujo general, se logra una interacción fluida entre todos los componentes del sistema, reduciendo el tiempo de respuesta.

4. **Adaptabilidad:**

- Este enfoque permite que el sistema se ajuste a diferentes tipos de consultas y bases de datos, garantizando respuestas relevantes independientemente de la naturaleza de la pregunta.

Conclusión

La integración del modelo **RAG (Zephyr)** mediante una API REST complementa de manera crucial el sistema desarrollado. No solo mejora la precisión de las respuestas, sino que también aprovecha al máximo la información estructurada y contextualizada obtenida de las bases de datos. Este flujo asegura que las consultas del usuario sean atendidas de manera eficiente, brindando respuestas que no solo resuelven dudas, sino que también proporcionan valor agregado al usuario.

Ejemplos de funcionamiento real

Respuesta del modelo: Year, 2013

Pregunta: what year is the game

Respuesta:

The game is set in the year 2013, as provided in the context information.

Pregunta: quienes son todos los hicieron el juego viticulture

Respuesta:

Los creadores del juego Viticulture son Alan Stone, Jamey Stegmaier, Beth Sobel, David Montgomery, y Jacqui Davis, también conocida como Christine Santana en algunas ediciones más recientes.

Pregunta: cuál es el objetivo

Respuesta:

El objetivo del juego es lograr 20 puntos de victoria (PV). Los puntos de victoria representan tu reputación como dueño del viñedo, y el jugador con más puntos al final del año en curso ganará. No hay límite de puntos, y en caso de empate se desempata por mayor cantidad de liras, seguido por valor total del vino de la bodega y finalmente por valor total de las uvas en los lagares.

Ejemplo de fallo real

Pregunta: cuáles son las reglas del juego

Respuesta:

La información de contexto proporciona algunas reglas específicas para una variante avanzada y una variante amistosa del juego, pero en general, las reglas del juego se desconocen sin más contexto. Sin embargo, se indica que se juega a Viticulture y se proporciona información sobre cómo se reparten los recursos iniciales y se colocan las fichas de gallo para determinar el orden de los jugadores. No se proporciona información sobre cómo se juegan las acciones en el juego, cómo se ganan puntos de victoria, o cómo se gana la partida en general.

4.3.6. Menú Principal

El **Menú Principal** del sistema fue diseñado con un enfoque en la simplicidad y funcionalidad, proporcionando una **interfaz por comandos** que facilita la interacción del usuario con el sistema.

La interfaz solicita al usuario que ingrese una consulta en lenguaje natural, lo cual desencadena automáticamente todos los procesos previamente descritos en las secciones anteriores. De esta manera, el usuario no necesita preocuparse por seleccionar bases de datos, tópicos o el tipo de consulta, ya que el sistema se encarga de procesar la entrada y devolver una respuesta precisa y contextualizada.

Funcionamiento del Menú Principal

1. **Ingreso de la consulta:**
 - El usuario introduce su pregunta o consulta en un campo de texto.
2. **Procesamiento automático:**
 - El sistema aplica de forma secuencial todos los módulos previamente detallados:
 - Clasificación de la consulta (Clasificador LLM, sección **4.3.3.3**).
 - Selección de la base de datos correspondiente (Data Base Selector, sección **4.3.3.5**).
 - Ejecución de la consulta específica según el tipo de base de datos seleccionada (sección **4.3.4. Query Dinámica**).
 - Generación de la respuesta utilizando el modelo RAG (sección **4.3.5. RAG**).
3. **Devolución de la respuesta:**
 - El sistema presenta al usuario una respuesta completa, procesada y personalizada según su consulta, acompañada de información adicional relevante cuando sea necesario.

Ventajas del Menú Principal

1. **Interfaz amigable:**
 - La sencillez de un sistema basado en comandos permite que usuarios con diferentes niveles de experiencia técnica puedan interactuar con el sistema sin dificultades.
2. **Automatización del flujo:**
 - Todo el proceso, desde la clasificación hasta la generación de la respuesta, se realiza de manera automática, minimizando la necesidad de intervención del usuario.
3. **Versatilidad:**
 - Al unificar todos los procesos en una única interfaz, el sistema es capaz de manejar consultas de diversa índole sin requerir ajustes manuales.
4. **Eficiencia:**
 - El usuario obtiene una respuesta rápida y precisa sin tener que preocuparse por detalles técnicos o procedimientos intermedios.

Conclusión

El **Menú Principal** actúa como el punto de entrada del sistema, ofreciendo una experiencia de usuario fluida y simplificada. Al combinar todos los componentes desarrollados en una única interfaz, el sistema garantiza que las consultas sean gestionadas de manera eficiente, dejando al usuario enfocado únicamente en formular sus preguntas y obtener respuestas útiles.

4.4. Ejercicio 2

En este ejercicio se desarrolló un agente basado en LlamaIndex que actúa como un intermediario entre el usuario y las bases de datos, utilizando el modelo Ollama como LLM. Este agente tiene como objetivo gestionar consultas en lenguaje natural, interpretarlas y utilizar herramientas específicas para buscar información en las bases de datos del sistema, aprovechando las funcionalidades previamente descritas en el Ejercicio 1.

El agente utiliza un enfoque modular en el que las herramientas (tools) implementadas, como graph_tool, tabular_tool y vec_tool, están diseñadas para interactuar directamente con los distintos tipos de bases de datos del sistema, llamando a las funciones de búsqueda mencionadas en la sección de Query Dinámica.

A continuación, se describe en detalle el funcionamiento del agente, las herramientas implementadas y cómo estas interactúan con el sistema.

4.4.1. Introducción

El agente desarrollado en este ejercicio se basa en **LlamaIndex**, una poderosa biblioteca para construir índices y agentes que interactúan con diversas fuentes de datos utilizando un modelo de lenguaje grande (**LLM**). En este caso, el modelo utilizado es **Ollama**, seleccionado por su capacidad para interpretar y responder consultas complejas en lenguaje natural.

Objetivo del Agente

El agente tiene como finalidad:

1. Interpretar la consulta del usuario.
2. Seleccionar la herramienta adecuada según el tipo de datos requerido.
3. Ejecutar la consulta utilizando las herramientas (**tools**) previamente implementadas.
4. Devolver una respuesta clara y precisa al usuario.

4.4.2. Herramientas del Agente (Tool)

En este apartado se describen las herramientas utilizadas por el agente, las cuales están diseñadas para gestionar de manera eficiente las consultas del usuario. Cada herramienta llama a los consultores descritos en los apartados 4.3.4.1. Consultar Base de Datos vectorial, 4.3.4.2. Consultar Base de Datos vectorial, y 4.3.4.3. Consultar Base de Datos de Grafo, dependiendo del tipo de consulta.

Estas herramientas son fundamentales para la modularidad y funcionalidad del sistema, ya que permiten integrar diferentes bases de datos y ejecutar consultas específicas según las necesidades del usuario.

1. *Vector Tool*

La herramienta **Vector Tool** está diseñada para llamar a la función previamente descrita en el apartado 4.3.4.1. que realiza consultas a las bases de datos vectoriales, donde se almacenan datos semánticos representados como vectores.

Funcionalidad:

- Realiza una búsqueda de contexto relevante en el espacio vectorial utilizando la consulta del usuario.
- Utiliza modelos LLM para enriquecer el análisis y refinar la consulta.
- Devuelve resultados basados en similitudes semánticas, ayudando a identificar información relacionada incluso cuando las palabras clave no coinciden exactamente.

2. *Tabular Tool*

La herramienta **Tabular Tool** está diseñada para llamar a la función previamente descrita en el apartado 4.3.4.2. para interactuar con bases de datos tabulares, como hojas de cálculo o tablas estructuradas en un DataFrame.

Funcionalidad:

- Interpreta la consulta del usuario para determinar qué fila y columna son relevantes.
- Utiliza un modelo LLM para analizar y comprender el contexto de la consulta.
- Realiza la búsqueda directamente en la base de datos tabular y devuelve el dato solicitado junto con información sobre su ubicación en la tabla (columna y fila).

3. *Graph Tool*

La herramienta **Graph Tool** está diseñada para llamar a la función previamente descrita en el apartado 4.3.4.4. e interactuar con bases de datos de grafos, que son útiles para modelar relaciones complejas entre nodos y entidades.

Funcionalidad:

- Clasifica la consulta para determinar qué tipo de información se busca en el grafo, como roles, personas, juegos o relaciones entre ellos.
- Identifica elementos específicos (nodos) mencionados en la consulta, como nombres de personas o juegos.
- Aplica filtros al grafo utilizando la librería **NetworkX** para extraer la información relevante y devuelve la respuesta en formato de texto.

Resumen de las Herramientas

Cada herramienta desempeña un papel único en el sistema, permitiendo que el agente gestione consultas específicas:

- **Tabular Tool** es ideal para datos estructurados y consultas directas.
- **Graph Tool** es esencial para explorar relaciones complejas y dependencias en grafos.
- **Vector Tool** permite buscar información basada en similitudes semánticas y conceptos abstractos.

Estas herramientas, trabajando en conjunto con el agente y las funcionalidades de clasificación y selección de bases de datos, aseguran un sistema robusto, modular y eficiente para resolver una amplia variedad de consultas del usuario.

4.4.3. Desarrollo

Funcionamiento General del Agente

El flujo de trabajo del agente es el siguiente:

- 1. Recepción de la consulta:**
 - El usuario ingresa una pregunta o consulta en lenguaje natural.
- 2. Clasificación inicial:**
 - El agente, con el soporte del modelo Ollama, analiza la consulta para determinar el tipo de información solicitada.
- 3. Selección de herramienta:**
 - Dependiendo del tipo de consulta, el agente selecciona una de las herramientas disponibles (graph_tool, tabular_tool o vec_tool) para realizar la búsqueda.
- 4. Ejecución de la consulta:**
 - La herramienta seleccionada llama a las funciones descritas en la Query Dinámica del Ejercicio 1, interactuando directamente con la base de datos correspondiente.
- 5. Generación de respuesta:**
 - El agente recopila los resultados de la herramienta utilizada y genera una respuesta en lenguaje natural para el usuario.

4.4.4. Resultados

Luego de un exhaustivo uso del agente, ajustando sus variables y modificando los prompts, se lograron identificar diversos comportamientos y resultados. Si bien en algunas ocasiones el agente proporcionó respuestas correctas, como en el caso en que se le solicitó el puntaje de un juego y, tras varios intentos fallidos, logró responder de forma válida, en la mayoría de las pruebas surgieron problemas significativos.

A continuación, se detallan los principales problemas detectados:

- **Clasificación incorrecta:** El agente, en ocasiones, no clasifica adecuadamente las consultas y redirige la pregunta a una base de datos errónea, ignorando las instrucciones de priorizar la base más relevante.
- **Modificación de la consulta:** A pesar de indicarle explícitamente que no debe alterar la query proporcionada, el agente frecuentemente la modifica, lo que afecta la precisión del resultado.
- **Ignorar respuestas válidas:** En múltiples casos, aunque el agente encuentra la respuesta correcta en la base de datos consultada, no la toma como válida y continúa buscando en otras bases, generando retrasos y respuestas erróneas.

TRABAJO PRÁCTICO N° 2
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA
PROCESAMIENTO DEL LENGUAJE NATURAL
ROSARIO - SANTA FE

- **Alucinación y sobrecarga:** Al incrementar la temperatura de creatividad del modelo LLM, el agente puede entrar en un loop de alucinaciones, generando respuestas incoherentes que terminan sobrecargando el buffer del equipo y bloqueando su funcionamiento.
- **Time Out:** Al no utilizar una gpu el agente no logra formular una respuesta en el tiempo estipulado y termina dando error.
- **Máximas Iteraciones:** A veces cuando no llega a alucinar y sobrecargar se detiene de manera inesperada en distintas iteraciones dando como resultado que “Error al procesar la consulta: Reached max iterations.”

Si bien se lograron algunos resultados satisfactorios, estos problemas recurrentes limitan la efectividad del agente y destacan áreas clave para futuras optimizaciones.

Ejemplo de respuesta correcta pero que le consulta a la base de datos errónea al principio ignorando las reglas dadas:

```
==== Ejemplo de interacción con el agente ReAct ====

Consulta 1: ¿Que puntaje tiene el juego?
> Running step 346c3a11-b4cf-4466-9396-d400b63fc174. Step input: ¿Que puntaje tiene el juego?
Thought: El usuario está preguntando sobre un juego y necesito saber su puntaje.
Action: graph_tool
Action Input: {'properties': AttributedDict({'query': AttributedDict({'title': 'Query', ('type': 'string'))}), 'required': ['query'], 'type': 'object'})
Observation: Error: graph_tool() got an unexpected keyword argument 'properties'
> Running step 6cccd82b-23f0-4f93-b8cb-083e480bb2d. Step input: None
Thought: El usuario estuvo un error al ejecutar la herramienta graph_tool.
Action: tabular_tool
Action Input: {'type': 'object'}
Observation: Error: tabular_tool() got an unexpected keyword argument 'type'
> Running step 7648c6df-681a-48b4-a299-9eb539b0d6d1. Step input: None
Thought: La herramienta tabular_tool no acepta el tipo de respuesta como parámetro, lo que indica que puede ser una tabla.
Action: graph_tool
Action Input: {'query': 'puntaje del juego'}
Clasificación: La pregunta "puntaje del juego" no se ajusta a ninguna de las categorías proporcionadas, ya que estas categorías están relacionadas con roles, personas y juegos, pero no con métricas de evaluación como el puntaje.
Observation: No se pudo clasificar la consulta. Intento reformular la pregunta.
> Running step 448ce092-44f7-4b76-8e92-c2b6d1aa7. Step input: None
Thought: La herramienta graph_tool no pudo entender la consulta del usuario, lo que sugiere que la pregunta puede ser ambigua o no tener suficiente contexto.
Action: tabular_tool
Action Input: {'query': '¿Cuál es el puntaje actual de un juego?'}
Respuesta del modelo: Avg. Rating, 7.486
Observation: {'Avg. Rating, 7.486'}
> Running step 2d2484b8-1dcf-4d46-854f-c82157a0d4db. Step input: None
Thought: La herramienta tabular_tool devolvió una respuesta con un valor promedio de rating para un juego.
Answer: El puntaje del juego es de 7.486 en promedio.
Respuesta 1: El puntaje del juego es de 7.486 en promedio.
```

```
==== Ejemplo de interacción con el agente ReAct ====

Consulta 1: cuanto pesa el juego
> Running step 614ec0b9-1d45-4952-8afc-3f48cb51bcb. Step input: cuanto pesa el juego
Thought: El usuario está preguntando por el peso del juego, pero no tengo información suficiente para proporcionar una respuesta precisa.
Action: graph_tool
Action Input: {'query': 'peso del juego', 'properties': AttributedDict({'title': 'Peso del juego', ('type': 'string'))})
Observation: Error: graph_tool() got an unexpected keyword argument 'properties'
> Running step 8e10b097-b55d-4b49-a3f1-b55ae54322d0. Step input: None
Thought: El usuario me informó que el parámetro 'properties' no es válido para la herramienta graph_tool.
Action: tabular_tool
Action Input: {'query': 'peso del juego', 'type': 'string'}
Observation: Error: tabular_tool() got an unexpected keyword argument 'type'
> Running step 9fa5528f-bb08-4a49-a508-d8ff05dc58e. Step input: None
Thought: El usuario me informó que el parámetro 'type' no es válido para la herramienta tabular_tool. Es posible que deba utilizar solo el parámetro 'query'.
Action: graph_tool
Action Input: {'query': 'peso del juego'}
Clasificación: La consulta "peso del juego" no se clasifica dentro de ninguno de los tipos de consultas que has proporcionado. Estos tipos de consultas se centran en la relación entre personas, roles y juegos, mientras que "peso" no se pudo clasificar la consulta. Intento reformular la pregunta.
> Running step a805de13b7-4b76-901e-22e63f97f1. Step input: None
Thought: El usuario me informó que no se pudo procesar su consulta para determinar el peso del juego. Es posible que deba reformular la pregunta de una manera diferente.
Action: tabular_tool
Action Input: {'query': '¿Cuál es el peso del juego en promedio?'}
Respuesta del modelo: Weight, 2.9
Observation: {'Weight, 2.9'}
> Running step 59f7e092-e1b6-41d9-a7c7-f954e26f0604. Step input: None
Thought: La herramienta tabular_tool devuelve un valor específico para el peso del juego, que es 2.9.
Answer: El juego pesa aproximadamente 2.9 unidades.
Respuesta 1: El juego pesa aproximadamente 2.9 unidades.
```

```
==== Ejemplo de interacción con el agente ReAct ====

Consulta 1: cual es la mecánica del juego
> Running step 614c1d9d-9e1c-4fb4-b4bb-098ad16a81d. Step input: cual es la mecánica del juego
Thought: El usuario está buscando información sobre el juego, pero no especifica qué juego en particular.
Action: graph_tool
Action Input: {'properties': AttributedDict({'query': AttributedDict({'title': 'Query', ('type': 'string'))}), 'required': ['query'], 'type': 'object'})
Observation: Error: graph_tool() got an unexpected keyword argument 'properties'
> Running step 76028460-a61a-44ce-8b82-c3bdab2424da. Step input: None
Thought: El usuario está experimentando un error al utilizar el tool graph_tool. Es posible que la sintaxis de la herramienta no sea compatible con el formato de entrada que estoy utilizando.
Action: tabular_tool
Action Input: {'query': '¿Cuál es la mecánica del juego?'}
Respuesta del modelo: columna. La pregunta "¿Cuál es la mecánica del juego?" no tiene una columna directa en la lista proporcionada. Sin embargo, si tengo que elegir una de las columnas es Error: la columna identificada no está en la lista de columnas.
Observation: None
> Running step 22da46aa-17af-afe8-9bd9-d883f1c51fd. Step input: None
Thought: El usuario no proporcionó una respuesta al utilizar el tool tabular_tool. Es posible que la herramienta no tenga información relevante para responder a la pregunta.
Action: vec_tool
Action Input: {'query': '¿Cuál es la mecánica del juego?'}
Observation: MECANICA DEL JUEGO Los jugadores distribuyen Los trabajadores y utilizan Los visitantes para completar varias tareas a lo largo de las estaciones (cada trabajador se coloca en una estación y realiza una tarea).
> Running step 9697960b-7fa3-4ff7-8f4f-cf5aed51cfc. Step input: None
Thought: El usuario proporcionó una respuesta detallada sobre la mecánica del juego, pero no especificó qué juego es específicamente.
Answer: El juego parece ser un juego de estrategia y planificación, donde los jugadores deben decidir cómo asignar a sus trabajadores para maximizar sus ganancias. La mecánica del juego es la asignación de recursos y la planificación.
Respuesta 1: El juego parece ser un juego de estrategia y planificación, donde los jugadores deben decidir cómo asignar a sus trabajadores para maximizar sus ganancias. La mecánica del juego es la asignación de recursos y la planificación.
```

Ejemplo de Error máximas iteraciones:

```
Consulta 1: Cuanto pesa el juego?
> Running step 83b69579-8d96-4457-ae08-77846f5ce8e2. Step input: Cuanto pesa el juego?
Thought: The user is asking about the weight of the game, but I'm not sure what kind of game it is or if it has a weight associated with it.
Action: graph_tool
Action Input: {'properties': AttributedDict({'query': AttributedDict({'title': 'Peso del juego'}, ('type', 'string')))), 'required': ['query'], 'type': 'object'}
Observation: Error: graph_tool() got an unexpected keyword argument 'properties'
> Running step 3440e215-6d5e-48be-a225-5d05945827ce. Step input: None
Thought: The 'graph_tool' tool does not have a 'properties' argument. I need to use the correct arguments for this tool.
Action: tabular_tool
Action Input: {'type': 'object'}
Observation: Error: tabular_tool() got an unexpected keyword argument 'type'
> Running step 28379597-e314-4abe-b41a-3a203c448597. Step input: None
Thought: The 'tabular_tool' tool does not have a 'type' argument. I need to use the correct arguments for this tool.
Action: vec_tool
Action Input: {'query': 'Peso del juego'}
Observation: DESCRIPCION viticulture es un juego de estrategia basado en la colocación de trabajadores. La Edición Esencial incluye la segunda edición de viticulture junto a varios elementos
> Running step 1e580e36-0942-47d7-98b4-df242f24683d. Step input: None
Thought: The game 'Viticulture' is a strategy game that involves placing workers, and the 'Edición Esencial' includes the second edition of Viticulture along with elements from the original.
Action: graph_tool
Action Input: {'query': 'Peso de la expansión Tuscany en el juego Viticulture'}
Observation: No se pudo clasificar la consulta. Intenta reformular la pregunta.
> Running step ef103102-84d5-4156-a91f-b7e4ae49acfe. Step input: None
Thought: The query could not be classified, so I need to rephrase the question to find more relevant information.
Action: vec_tool
Action Input: {'query': '¿Cuál es el peso de la expansión Tuscany en el juego Viticulture?'}
Observation: DESCRIPCION viticulture es un juego de estrategia basado en la colocación de trabajadores. La Edición Esencial incluye la segunda edición de viticulture junto a varios elementos
Respuesta 1: Error al procesar la consulta: Reached max iterations.
```

4.4.5. Conclusiones

Más allá de las respuestas erróneas mostradas anteriormente podemos destacar del modelo lo siguiente:

Ventajas del Uso de LlamaIndex y Ollama

1. Adaptabilidad:

- LlamaIndex permite integrar diferentes fuentes de datos y personalizar la interacción entre el agente y estas fuentes.

2. Capacidad de Interpretación:

- Ollama destaca por su capacidad de comprender consultas complejas y ambiguas, asegurando respuestas precisas y contextualizadas.

3. Modularidad:

- Las herramientas implementadas permiten que el sistema sea extensible y fácil de mantener.

4. Automatización:

- El agente automatiza completamente el flujo de consultas, reduciendo la intervención manual del usuario y optimizando el tiempo de respuesta

Desventaja

Además de lo mencionado también se concluye que la mayor desventaja del agente es la gran capacidad de alucinar y no solo dar respuestas erróneas sino que también perder respuestas que había obtenido como buenas pero que decidió descartarlas.

5. Conclusiones Generales

5.1. Conclusiones Generales

Se logró cumplir con el objetivo principal del proyecto: entender el funcionamiento y manejo de herramientas modernas como bases de datos tabulares, grafos, vectoriales, agentes y modelos de lenguaje. A pesar de las limitaciones, se alcanzó una integración funcional que permite consultas usuario-base de datos de manera fluida y comprensible.

Se desarrolló una aplicación que, aunque no es completamente óptima, demuestra el potencial de un sistema de consultas dinámicas y adaptativas, donde el usuario puede interactuar con diferentes tipos de bases de datos utilizando un único punto de acceso centralizado. Este avance resulta significativo como ejercicio práctico, mostrando cómo estas herramientas pueden trabajar en conjunto para resolver problemas complejos y estructurados.

5.2. LLM utilizadas

En el desarrollo de este proyecto se utilizaron tres modelos de lenguaje principales: **Gwen**, **Zephyr**, y **Ollama**, cada uno con un propósito específico dentro del sistema. A continuación, se detallan sus roles, características principales y una conclusión general sobre su desempeño.

Gwen

Gwen fue el modelo base encargado de interpretar las consultas de los usuarios y clasificarlas en categorías específicas. Este modelo destacó por su capacidad para realizar tareas de clasificación de manera eficiente y adaptarse a las necesidades de un flujo dinámico. Además, permitió identificar palabras clave en las consultas y asociarlas con las herramientas o bases de datos adecuadas.

Sin embargo, Gwen mostró limitaciones en la comprensión de consultas más ambiguas o mal formuladas, lo que en algunos casos derivó en clasificaciones incorrectas o inconsistentes. Para mejorar su desempeño, sería necesario optimizar los prompts y proporcionar datos de entrenamiento más variados y específicos.

Zephyr

Zephyr fue utilizado como el motor detrás del sistema **RAG** (Retrieve and Generate). Este modelo tuvo la tarea de recibir el contexto extraído de las bases de datos junto con la consulta del usuario y generar una respuesta precisa. Su integración mediante una API permitió un despliegue rápido y eficiente.

Zephyr mostró fortalezas al combinar información estructurada con capacidades generativas, especialmente en consultas complejas donde era necesario un análisis más profundo. No obstante, su dependencia del contexto inicial afectó su precisión cuando los datos proporcionados eran limitados o incompletos, lo que expuso la importancia de mejorar la extracción y selección de contexto.

Ollama

Ollama fue utilizado como el modelo LLM central en la implementación del agente en el **Ejercicio 2**. Este modelo fue integrado con **LlamaIndex**, y su principal función fue operar como núcleo para coordinar las herramientas del sistema (Graph_Tool, Tabular_Tool y Vec_Tool) y generar respuestas adaptadas a las consultas del usuario.

Ollama demostró ser un modelo flexible y adecuado para interactuar con múltiples herramientas de manera simultánea, mostrando un buen equilibrio entre capacidad de generación y procesamiento de contexto. Sin embargo, su desempeño dependió en gran medida de la calidad de las herramientas subyacentes y del flujo de consultas, destacándose la necesidad de mejorar la integración entre el modelo y dichas herramientas.

Impacto del Idioma en las Respuestas

Durante las pruebas realizadas para este trabajo práctico, se observó que los modelos de lenguaje mostraron un desempeño significativamente mejor cuando se les proporcionaban consultas en inglés, en comparación con consultas en español. Esto se debe a que muchos modelos de LLM han sido entrenados con una mayor cantidad de datos en inglés, lo que les permite interpretar y generar respuestas con mayor precisión en este idioma.

En contraste, al trabajar en español, los resultados fueron menos consistentes, con respuestas más propensas a errores o ambigüedades, especialmente en consultas complejas o contextos donde se requería precisión. Esta limitación representa un desafío para el desarrollo de aplicaciones en español y pone de manifiesto la importancia de seguir entrenando y optimizando modelos de lenguaje para este idioma. Para futuros trabajos enfocados en usuarios de habla hispana, sería necesario explorar modelos que estén específicamente diseñados o afinados para el idioma español, así como mejorar los corpus de datos en este idioma.

Conclusión General de los Ilm

El uso combinado de estos tres modelos permitió construir un sistema que integraba capacidades de clasificación, generación y recuperación de información de forma efectiva. Cada modelo cumplió un rol específico dentro del sistema, logrando trabajar en conjunto para procesar consultas y devolver respuestas útiles.

Sin embargo, una conclusión importante es que el desempeño de los LLM depende fuertemente del diseño de los prompts, la cantidad y calidad de los datos disponibles, y la integración con otras herramientas. Los resultados obtenidos muestran que, aunque los LLM tienen un gran potencial, requieren un refinamiento constante para alcanzar niveles óptimos de precisión y consistencia. En proyectos futuros, sería interesante explorar modelos más especializados o de mayor capacidad para abordar las limitaciones identificadas.

5.3. Problemáticas Encontradas

Durante el desarrollo del proyecto se identificaron varias dificultades:

1. **Cantidad de datos limitada:**

Las bases de datos utilizadas no tenían un volumen o diversidad suficiente para realizar consultas avanzadas o detectar patrones complejos. Esto limitó la capacidad de probar escenarios reales más amplios.

2. **Dificultades en la creación de prompts efectivos:**

Los modelos de lenguaje utilizados (LLMs) requieren instrucciones precisas para devolver resultados correctos. Encontrar la estructura óptima para los prompts fue un desafío constante.

3. **Resultados inexactos del agente:**

Hubo casos donde el agente devolvió respuestas incorrectas o imprecisas. Esto ocurrió por errores en el flujo de clasificación, limitaciones en el entendimiento del contexto por parte del modelo LLM, o por falta de datos en las bases de datos para respaldar la consulta.

4. **Interrupciones en el flujo de respuesta:**

En algunos casos, el proceso se desviaba y no llegaba a la respuesta correcta, lo que afecta la confiabilidad del sistema en situaciones más complejas.

5. **Requerimiento de hardware:**

Para que el agente pueda dar una respuesta en tiempo y forma se tuvo que utilizar la pc con GPU de google colab. El inconveniente de esto es que el mismo tiene una limitación de tiempo de uso.

6. **Limitación en las consultas de grafo al usar network x:**

Este enfoque tiene ciertas desventajas:

1. **Menos eficiente:** Las funciones programáticas pueden ser más lentas que un motor de base de datos optimizado para grafos.
2. **Falta de sintaxis declarativa:** Tienes que escribir el código exacto para realizar cada consulta, lo cual puede ser más complejo que usar un lenguaje como Cypher.
3. **No es reutilizable como un lenguaje de consultas estándar.** como Cypher (de Neo4j) o SQL

5.4. Mejoras a Futuro

Si bien se logró una base funcional, existen varias áreas que podrían mejorarse para optimizar el sistema y hacerlo más robusto:

1. **Optimización de Prompts:**

Diseñar prompts más específicos y adaptables para mejorar la interacción con los modelos de lenguaje. Esto incluye la creación de plantillas dinámicas para distintos tipos de consultas.

2. **Ampliación de Bases de Datos:**

Construir bases de datos más completas y diversas para simular escenarios reales y proporcionar mejores resultados al usuario. Esto incluye agregar más datos tabulares, ampliar las relaciones en el grafo y crear una base vectorial más rica.

3. **Mejoras en la Interfaz de Usuario:**

Implementar un menú más interactivo e intuitivo que permita al usuario visualizar los resultados de manera más clara, como gráficos o resúmenes visuales.

4. **Aumento de la Precisión del Agente:**

Trabajar en el flujo de clasificación y en la integración de las herramientas para reducir errores y garantizar que las respuestas sean correctas. Esto podría incluir el uso de agentes más avanzados o el ajuste fino de los modelos existentes.

5. **Automatización de la Selección de Contexto:**

Mejorar la manera en que se extrae el contexto relevante para las consultas, logrando que las respuestas sean más precisas y ajustadas al objetivo del usuario.

Finalmente, aunque no hubo tiempo suficiente para crear bases de datos más grandes o implementar una solución completamente optimizada, los resultados obtenidos muestran que el sistema tiene un potencial significativo para escalar y mejorar en el futuro. Este ejercicio ha sentado una base sólida para desarrollar sistemas de consultas inteligentes y prácticas.

6. Anexo

Video de la reseña utilizada para la base de datos:

https://www.youtube.com/watch?v=ysSOgQQKIU&ab_channel=TableTopBunnyOficial