



UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Lenguaje Natural

Informe sobre Trabajo Práctico Integrador

2024

Francisco Domingo

Índice

Introducción	3
Ejercicio 1: Chatbot experto en Rajas of the Ganges	3
Ejercicio 2: Agente	7
Conclusiones	8

Introducción:

En este trabajo práctico nos proponemos implementar múltiples herramientas y conceptos aprendidos durante el cursado de Procesamiento de Lenguaje Natural con el fin de crear un chatbot y un agente especializados en responder preguntas sobre un juego de mesa. El juego asignado es “Rajas of the Ganges”, y para nutrir a nuestras bases de datos, usaremos cuatro diferentes fuentes de información:

- La página web “boardgamegeek.com”, que condensa data de todo tipo sobre múltiples juegos similares a éste.
- Guías rápidas de juego desarrolladas por la comunidad.
- Reviews y Gameplays en formato video de la plataforma YouTube.
- Wikidata, como una base de datos de grafos.

Ejercicio 1:

Considerando que nuestro chatbot debe funcionar tanto en inglés como en español, tomé la decisión de generar todo mi backend en inglés (idioma semántica y gramaticalmente más sencillo), y usar un moderador de lenguaje para hacer las traducciones en caso de ser necesario. Esto realmente facilita la implementación, además de mejorar la performance de los modelos de LLM.

Para generar mis bases de datos, lo primero que realizamos es generar una fuente de datos en formato *string*, consumiendo tanto las guías rápidas de juego desarrolladas por la comunidad de Board Game Geek, como también dos videos de YouTube. Por la consideración anterior, todas estas fuentes se encuentran en inglés.

Por otro lado, hice WebScrapping con Selenium sobre la página web ya mencionada, con el objetivo de recolectar diferentes datos y estadísticas sobre el juego: puntaje, cantidad de jugadores, cantidad de compradores, etc.

Finalmente, utilicé la base de datos de grafos provista por Wikipedia para poder relacionar el juego con sus desarrolladores y/o distribuidores. Usando queries de SPARQL puedo obtener datos actualizados sobre los juegos relacionados a “Rajas of the Ganges” que me permitan hacer recomendaciones específicas para el usuario.

A partir de estas fuentes, generé los chunks, la metadata y los ids necesarios para finalmente embeberlos y así alimentar una base de datos ChromaDB.

Para estos embeddings, utilicé all-MiniLM-L6-v2. Por lo que pude investigar, este modelo es una opción ideal para generar embeddings porque combina buena precisión con alta

eficiencia. Es rápido, liviano y genera vectores de baja dimensionalidad (384), lo que lo hace perfecto para proyectos con pocos recursos o donde se necesita procesar textos variados. Además, fue entrenado con datos muy diversos y multilingües, así que funciona bien en un montón de situaciones. Si bien hay modelos más precisos para casos muy específicos o si tenés mucha capacidad computacional, este es una gran elección si buscás algo rápido, versátil y eficiente.

En la metadata *type* guardé uno de estos cuatro valores: *rules*, *review*, *stats* y *related_games*. Las guías rápidas caían en la primera categoría, mientras que los videos de youtube, en la segunda. Las estadísticas en *stats*, y el grafo en *related_games*.

Esta clasificación nos ayudará más tarde para las consultas en el prompt. Cada prompt es clasificado en alguno de estos tipos, a través de dos modelos diferentes: un RandomForest y un LLM usando la librería HuggingFace.

Entiendo que el modelo de LLM utilizado (Qwen/Qwen2.5-72B-Instruct) es ideal para tareas de clasificación complejas gracias a su enorme capacidad de comprensión, con 72 mil millones de parámetros que le permiten captar detalles y matices que otros modelos más chicos no logran. Está diseñado específicamente para seguir instrucciones de manera precisa, lo que lo hace perfecto para prompts complejos y variados.

Para decidir qué algoritmo es mejor (RF vs LLM), generé mock data usando ChatGPT, y tomé como parámetro el accuracy. Mientras que random forest acertaba el 55% de las veces, el modelo de LLM lo hacía un 72% de sus ejecuciones. Por esto mismo es que, como clasificador, elegí el LLM.

Finalmente, configuré un retriever, cuyo objetivo es traer la información de la fuente más relevante según el prompt introducido. Aquí usamos el clasificador LLM y un reranker para la base de datos vectorial.

Después de mucha experimentación y prompt engineering, me quedé con el siguiente input.

```
def prompt_prep(context, user_input):
    query = f"""You're helping an end user with a board game called Rajas of the Ganges.
    You're a subject matter expert that will provide an answer to, ideally, a single question from this user.
    Your answer has to be as much direct and consistent as possible.
    If the context available is not enough to answer the question, ask the end user to reformulate the question.
    This is the context available:
    {context}
    This is the end user question:
    {user_input}
    """
    return query
```

Por momentos, el motor del LLM devolvía letras en japonés y caracteres extraños.

```

rules
rules
rules
! protection.驾驶证 warnings 다}')
连接, filters
执行 security) pay6 grand aepwization,
rules
rules

```

La razón era que el classifier estaba devolviendo cosas extrañas, producto de haberle instruido en “no agregar caracteres ni ningún otro contexto a las respuestas”. Mi objetivo era evitar salidas que escapen de los cuatro tipos antes definidos, pero como pude verificar, no es provechoso dar instrucciones en formato negativo.

Además, agregué a la función un arreglo bastante artesanal para asegurarme de que siempre devuelva una clase concreta.

```

def llm_classifier(prompt):
    message=[{"role": role, "content": content+prompt}]
    completion = client.chat.completions.create(model="Qwen/Qwen2.5-72B-Instruct", messages=message, max_tokens=200)
    classification = completion.choices[0].message.content.strip()
    if str(classification) not in types:
        print(f"Random classification, llm clasified {classification}.")
        classification = types[len(prompt)%3]
    print(classification.lower())
    return classification.lower()

```

Haciendo las modificaciones del prompt, prácticamente nunca usa el random classification, y la decisión siempre es tomada por el LLM.

Los resultados de nuestro chatbot son bastantes buenos, tanto en inglés como en español.

```

[69] help()
Rajas of the Ganges - HELP!
Ask me something: ¿Cuál es el puntaje promedio para este juego?
Así que hablas español. Deja que me preparo para la ocasión. *se prepara un mate*
Mi respuesta:
La calificación promedio para el juego "Rajas of the Ganges" es 7.731, basada en 14,702 calificaciones.

[67] help()
Rajas of the Ganges - HELP!
Ask me something: what is the average rating for this game?
Let me think...
This is my answer:
The average rating for Rajas of the Ganges is 7.731.

```

```
[65] help()

Rajas of the Ganges - HELP!
Ask me something: I really enjoyed this game. Which other board game do you recommend?
Let me think...
This is my answer:
I'm glad you enjoyed "Rajas of the Ganges"! Given the games from the same developer, you might also enjoy:

1. **Village** - A game that involves building and expanding a village, which can offer a similar strategic and thematic experience.
2. **Burg der 1000 Spiegel** - This game features a mix of strategy and puzzle-solving, which might appeal to you if you enjoyed the strategic e
3. **Monster Trap** - If you like games with a bit of a twist and strategic planning, this game might be a good fit.

Alternatively, from the same publisher, you might enjoy:

1. **YINSH** - A strategy game that involves capturing your opponent's pieces and forming lines, which can be a great match if you enjoy deep str
2. **Ta-Ke** - Another strategic game that focuses on capturing and placing pieces, similar to the strategic depth found in "Rajas of the Ganges.

I hope you find these recommendations helpful! Let me know if you have any other questions or need more suggestions.
```

```
help()

Rajas of the Ganges - HELP!
Ask me something: ¿Cuántos jugadores necesito para iniciar una partida?
Así que hablas español. Deja que me preparo para la ocasión. *se prepara un mate*
Mi respuesta:
Para empezar una partida de Rajas of the Ganges, necesitas al menos 2 jugadores. El juego admite hasta 4 jugadores.

Si juegas con 2 jugadores, utiliza el lado del tablero para 2 jugadores. Si juegas con 3 o 4 jugadores, utiliza el lado del tablero para 3 o 4 ju

Asegúrate de que los elefantes en la parte inferior derecha del tablero estén orientados en la dirección correcta para la cantidad de jugadores c

¡Avísame si tienes alguna otra pregunta!
```

Ejercicio 2:

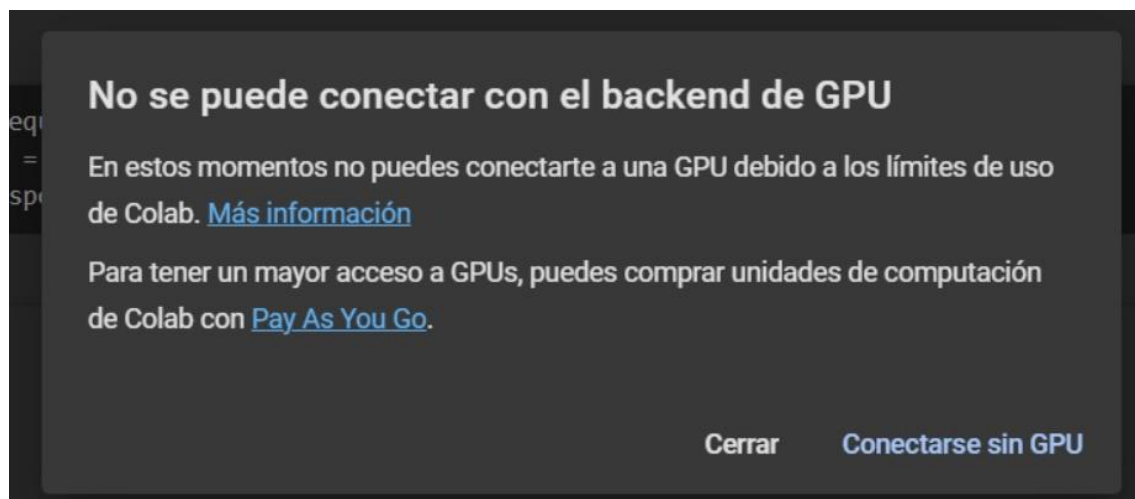
En este otro caso, implementé un sistema basado en un agente de inteligencia artificial diseñado para asistir a usuarios con preguntas relacionadas al juego antes mencionado.

Utilicé el modelo llama3.2, configurado con parámetros como un bajo nivel de aleatoriedad (`temperature=0.1`) para respuestas más consistentes y una ventana de contexto amplia (`context_window=4000`) para manejar consultas complejas. El agente usa las funciones ya definidas (`retriever_vector_db`, `retriever_dataframe_db` y `retriever_graph_db`) para extraer información relevante nuestras bases de datos.

Se utiliza la arquitectura ReAct (Reasoning + Acting), que permite que el modelo analice la consulta, decida qué herramienta usar y actúe en consecuencia. Esto se estructura bajo un formato claro que guía tanto al modelo como a los desarrolladores para mantener consistencia en el flujo de pensamiento, acción y observación.

El agente recibe la consulta del usuario, analiza cómo responder basándose en el contexto y utiliza las herramientas configuradas para encontrar la información adecuada. Si hay un error o una consulta inválida, se maneja la excepción de manera controlada para evitar fallos en el programa.

El código provisto corrió una única vez, luego de reiteradas re-instalaciones de librerías por problemas de incompatibilidades y de ejecución de Ollama sobre GoogleColab. Luego de esa ejecución (cambiando el entorno a tp4), el uso de GPU se acabó para mi cuenta, obteniendo este mensaje de error:



En consecuencia, la ejecución sin GPU finalizó siempre en timeout, incluso aumentando el parámetro a 500 o 1000 segundos. Lamentablemente, no pude probar en profundidad la calidad del código provisto, ni tampoco pude experimentar con las configuraciones del prompt para mejorar la salida.

Conclusiones

En este trabajo, implementamos herramientas de procesamiento de lenguaje natural para crear un chatbot y un agente especializado en el juego de mesa "Rajas of the Ganges". A través de diversas fuentes de datos y técnicas avanzadas, como embeddings generados con el modelo all-MiniLM-L6-v2 y consultas SPARQL para bases de datos de grafos, logramos construir un sistema robusto que funciona tanto en inglés como en español. Opté por el modelo de LLM, debido a su alta precisión y capacidad para manejar consultas complejas, superando a alternativas como Random Forest en clasificación. Los resultados obtenidos fueron positivos, destacándose la consistencia en las respuestas. Este trabajo evidencia el potencial del NLP para construir asistentes inteligentes en dominios especializados con contexto.