



Procesamiento del Lenguaje Natural
Tecnicatura universitaria en Inteligencia Artificial

2024

Docentes:

- Juan Pablo Mason
- Alan Geary
- Constantino Ferrucci
- Dolores Solberger

Alumno:

- Facundo Fontela (Legajo F-3724/9)

Introducción

En el siguiente informe se detallará el proceso, las problemáticas y la discusión de los resultados obtenidos en la implementación de un RAG y un agente especializados en el juego de mesa “Las Ruinas Perdidas de Arnak”.

Para ello se realizó un trabajo de extracción de información de distintas fuentes relacionadas con el juego en cuestión. Se conformaron 3 bases de datos distintas, vectorial, grafo y tabular, para consultar y dar contexto a los distintos modelos.

Además se implementaron 2 tipos distintos de clasificadores para determinar a qué base de datos consultar el contexto.

Las consultas de contexto, tanto para el grafo como la base tabular, se realizan de forma automática mediante el uso de un modelo de lenguaje pre entrenado.

Ejercicio 1)

Resumen

El objetivo del punto 1 es la implementación de un chatbot usando la técnica de RAG especializado en el juego “Las Ruinas Perdidas de Arnak”. En este punto se cubren distintos aspectos como: la extracción de datos y creación de las bases correspondientes, la implementación de los 2 clasificadores mencionados en la introducción, el desarrollo de los 3 retrieves de información correspondiente a cada fuente y finalmente el chatbot que hace uso de lo mencionado para alimentar un LLM pre entrenado y responder de acuerdo a la consulta recibida.

Desarrollo

Fuentes de datos

Base vectorial:

Se recopilaron manuales en formato pdf y en formato txt y se extrajo la transcripción de distintos videos de youtube de explicaciones y reglas de juego aportados por la comunidad.

Se realizó una limpieza del texto (obtenido todo en idioma español), utilizando librerías como nltk y regular expresiones. Se realizó un split del mismo en chunks, utilizando librería LangChain, y se almacenó en una base en memoria chromadb.

Grafo:

Mediante web scraping (utilizando la herramienta Selenium debido a que la página web es de contenido dinámico) se extrajo de BGG (board geek game) las distintas relaciones del juego con creadores, artistas, desarrolladores, publicadores, etc, y otros juegos en los que hayan participado.

Se generó un grafo con la librería rdflib de python, el cual es posible de consultar mediante el lenguaje sparql.

Base tabular:

De la misma página principal BGG se extrajeron todos los datos de carácter tabular, como pueden ser duración de partida, cantidad de jugadores, fechas relevantes, etc. Estos se agruparon en una estructura de (tipo, atributo, valor). El tipo sirve para reducir la cantidad de búsqueda (siendo por ejemplo 'jugadores' o 'fechas'), la columna atributo sirve para buscar la fila que más esté relacionada con la consulta, el valor contiene la información pertinente.

Clasificadores

Clasificador Logístico:

Para el desarrollo de este clasificador fue necesario generar un dataset de entrenamiento, para esto se generaron gran cantidad de prompts utilizando Chat GPT. Estos prompts identifican una posible consulta a la base vectorial (reglas de juego), al grafo (relaciones del juego) o a la base tabular (estadísticas en general).

Se entrenó un clasificador clásico, basado en regresión logística. Este arrojó resultados decentes para la poca cantidad de datos y el desbalance que presenta el dataset de entrenamiento., logrando buenas métricas de precisión, pero con malos resultados en recall o f1-score.

```
Precisión Regresión Logística: 0.8392857142857143
Reporte de clasificación Regresión Logística:
      precision    recall  f1-score   support

 grafo           1.00      0.33      0.50         6
 tabular          1.00      0.58      0.74        12
 vectorial        0.81      1.00      0.89        38

 accuracy                   0.84         56
 macro avg           0.94      0.64      0.71         56
 weighted avg          0.87      0.84      0.82         56
```

Clasificador basado en LLM:

Para el desarrollo de este clasificador se utilizó la librería hugging face con el objetivo de hacer uso del modelo 'qwen2.5', el cual es un modelo pre entrenado con 72b parámetros.

La estrategia utilizada fue: generar un prompt con el contexto necesario para que el modelo clasifique el prompt en alguna de las 3 clases correspondientes (fuentes de datos).

Se obtuvieron buenos resultados en preguntas típicas:

```
# Ejemplo 1: grafo
predicted_label = classify_text("¿Qien es el publicador de Las Ruinas perdidas Arnak?")
print(f"El texto es clasificado como: {predicted_label}")

El texto es clasificado como: grafo

[ ] # Ejemplo 2: vectorial
predicted_label = classify_text("¿Cómo hay que ordenar el tablero al principio?")
print(f"El texto es clasificado como: {predicted_label}")

El texto es clasificado como: Vectorial

[ ] # Ejemplo 3: tabular
predicted_label = classify_text("Cuantos jugadores pueden jugar?")
print(f"El texto es clasificado como: {predicted_label}")

El texto es clasificado como: tabular
```

Es difícil medir métricas en el clasificador, ya que se deben realizar gran cantidad de llamadas al modelo via API. Para poder dar claridad acerca del rendimiento de este clasificador se ejecutó el mismo sobre el conjunto de test utilizado en el clasificador logístico para medir la precisión en cada clase. Los resultados se muestran a continuación. Se puede observar un gran rendimiento en 'vectorial' y 'tabular', en grafo acierta la mitad de las veces.

```
[ ] accuracy_graph = results["grafo"]["correctos"]/results['grafo']['total']
print(f"Precision para clase grafo: {accuracy_graph}")

Precision para clase grafo: 0.5

accuracy_vectorial = results["vectorial"]["correctos"]/results['vectorial']['total']
print(f"Precision para clase vectorial: {accuracy_vectorial}")

Precision para clase vectorial: 0.7894736842105263

[ ] accuracy_tabular = results["tabular"]["correctos"]/results['tabular']['total']
print(f"Precision para clase tabular: {accuracy_tabular}")

Precision para clase tabular: 0.9166666666666666
```

Retrievers

Retriever grafo:

Para poder recuperar contexto del grafo primero fue necesario crear un esquema del mismo, para luego poder aportar a un modelo. Esto es fácilmente realizable mediante queries sparql que recuperan clases y relaciones.

Una vez hecho eso se creó una función que recibe el esquema y el prompt del usuario para, por medio del LLM (mismo modelo que el clasificador) se genere una query sparql dinámica que consulte en el grafo la información correspondiente.

Se obtienen resultados diversos. El modelo sabe generar consultas sparql y devolverlas correctamente de acuerdo al esquema, sin embargo hay situaciones donde, solo cambiando una letra o una mayúscula, la query generada es incorrecta (no devuelve ternas).

Un ejemplo de correcto funcionamiento:

```
# Ejemplo de retrieve del grafo en base a un prompt

retrieve_grafo("Quiénes publicaron el juego Las Ruinas Perdidas de Arnak?")

["(rdflib.term.URIRef('http://example.org/Czech%20Games%20Edition'),)",
 "(rdflib.term.URIRef('http://example.org/Br%C3%A4dspele.se'),)",
 "(rdflib.term.URIRef('http://example.org/Cranio%20Creations'),)",
 "(rdflib.term.URIRef('http://example.org/Devir'),)",
 "(rdflib.term.URIRef('http://example.org/DiceTree%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Fantasmagoria'),)",
 "(rdflib.term.URIRef('http://example.org/GaGa%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Games4you'),)",
 "(rdflib.term.URIRef('http://example.org/G%C3%A4m%20Klub%20Kft.'),)",
 "(rdflib.term.URIRef('http://example.org/HeidelB%C3%84R%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Hobby%20Japan'),)",
 "(rdflib.term.URIRef('http://example.org/IELLO'),)",
 "(rdflib.term.URIRef('http://example.org/Kaissa%20Chess%20%26%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Lautapelit.fi'),)",
 "(rdflib.term.URIRef('http://example.org/Lex%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Lord%20of%20Boards'),)",
 "(rdflib.term.URIRef('http://example.org/MINDOK'),)",
 "(rdflib.term.URIRef('http://example.org/More%20Fun%20Co.%2C%20Ltd.'),)",
 "(rdflib.term.URIRef('http://example.org/One%20Moment%20Games'),)",
 "(rdflib.term.URIRef('http://example.org/Rebel%20Sp.%20z%20o.o.'),)",
 "(rdflib.term.URIRef('http://example.org/Spilbr%C3%A6t.dk'),)",
 "(rdflib.term.URIRef('http://example.org/White%20Goblin%20Games'),)"]
```

Retriever tabular:

De forma muy similar al grafo, se aporta al LLM información de los tipos existentes en la base tabular para que este indique cuál le parece más apropiado para recuperar contexto (de alguna forma funciona como clasificador).

Una vez obtenido el tipo se realiza un filtrado mediante pandas y se devuelven las filas correspondientes.

Un ejemplo del funcionamiento:

```
[71] # ejemplo de retrieve tabular
      retrieve_tabular("Cuantos jugadores pueden jugar al juego?")

jugadores
['cantidad minima de jugadores: 1',
 'cantidad maxima de jugadores: 4',
 'cantidad de jugadores: 1 a 4',
 'cantidad recomendada de jugadores: 3']
```

Retrieve base vectorial:

Para devolver documentos de la base vectorial, en primera instancia se hace un embedding del prompt del usuario, para que chromadb devuelva los 15 documentos con semántica más similar.

A su vez se utiliza la librería rank_bm25 para realizar una búsqueda en base a palabras claves, recuperando así otros 10 documentos

Finalmente se aplica un rerank basado en scores que reordena los documentos utilizando un modelo de encoder mas preciso ([cross-encoder/ms-marco-MiniLM-L-6-v2](#)). Finalmente se devuelven los 5 mejor rankeados.

Un ejemplo del funcionamiento es el siguiente (donde la respuesta se ve truncada, solo se visualiza el primer documento devuelto):

```
print(retrieve_vectorial("¿Como funciona la mecanica de excavar un yacimiento"))
['de tus arqueólogos para que excave en cualquiera de los 5 yacimientos . también podrás excavar en los yacimientos y en cuanto se hayan descubierto. par
```

Retriever combinado:

Finalmente se crea una función que sirve de entrada para el RAG, combinando el clasificador con los distintos retrievers.

Esta recibe el prompt del usuario, se encarga de llamar al clasificador para obtener la fuente de datos pertinente y llama al retriever correspondiente.

RAG

Una vez implementado los pasos anteriores el RAG es muy simple. Es una función que únicamente recibe el prompt del usuario, llama al retriever combinado para recibir el contexto y finalmente llama al modelo gwen. Haciendo uso de prompt engineering se aporta el contexto y se pide responder a la consulta solicitada.

A continuación se muestran algunos ejemplos del mismo:

PROMPT: "Como se dispone el tablero inicialmente?"

Respuesta: Para disponer el tablero inicialmente, sigue estos pasos:

1. Despliega el tablero central por la cara deseada (puedes elegir entre el modo normal o el modo complejo).
 2. Coloca tu tablero de arqueólogo en tu espacio de jugador.
 3. Coloca tu arqueólogo en la casilla del yacimiento que hay en el tablero central.
 4. Si el yacimiento tiene un ídolo, quédatelo y resuelve de inmediato el efecto del ídolo. Si el yacimiento tiene dos ídolos, quédate los dos.
 5. Toma la loseta superior de la pila que se corresponda con el yacimiento y colócala en el tablero boca arriba.
 6. El yacimiento ahora está descubierto y su efecto se resolverá de inmediato.
- !

PROMPT: "Quienes publicaron el juego Las Ruinas Perdidas de Arnak?"

Respuesta: Las Ruinas Perdidas de Arnak fue publicado por **Czech Games Edition**. Esta compañía es conocida por publicar juegos de mesa de alta calidad, incluyendo títulos de tipo euro game. Sin embargo, es importante verificar esta información, ya que la lista proporcionada no especifica directamente esta publicación. Si necesitas más detalles o confirmación, no dudes en preguntar.

PROMPT: "Cuanto dura una partida normalmente?"

Respuesta: Una partida normalmente dura entre 30 y 120 minutos.

PROMPT: "Que hace un jugador en cada turno?"

Respuesta: 'En cada turno, el jugador activo escogerá una de las acciones disponibles: excavar en un yacimiento, conseguir recursos, habilidades y puntos. Además, durante tu turno solo podrás hacer una acción principal y tantas acciones gratuitas como desees.'

PROMPT: "Cuando finaliza la ronda?"

Respuesta: 'Una ronda finaliza cuando todos los jugadores han pasado, excepto al terminar la quinta ronda.'

PROMPT: "Que sucede en la quinta ronda?"

Respuesta: 'La partida finaliza al término de la quinta ronda, procediéndose a la evaluación final en la que cada jugador anota los siguientes puntos.'

Conclusiones

El RAG implementado cuenta con las herramientas necesarias para responder distintos tipos de preguntas, sabiendo consultar las distintas fuentes según sea el caso. El clasificador muestra buenos resultados en general, fallando principalmente en las consultas más relacionadas con el grafo.

Con respecto a las queries dinámicas se observa un buen resultado en la base tabular, mientras que la query dinámica sparql tiende a dar resultados diversos, fallando en algunos casos en la construcción de la misma. Una posible solución a este problema es utilizar lenguajes de consulta más simples como cypher, o utilizar

modelos con finetuning específicos para generar consultas a grafos, los cuales existen pero en general no son de libre acceso.

Cabe destacar que los resultados obtenidos al consultar prompts relacionados con la base vectorial son pertinentes y de acuerdo al juego.

El modelo de lenguaje utilizado para el RAG, clasificador y retrievers es Qwen2.5. Se probaron distintos modelos open source, como zephyr y mistral, sin embargo el elegido demostró tener un mejor desempeño generando queries dinámicas y respondiendo concretamente, sin imaginar respuestas.

Finalmente, el RAG, cumple con los objetivos propuestos y mostrando un rendimiento aceptable, fallando en algunas situaciones particulares donde el retriever del grafo no genera bien la consulta sparql, o en casos donde el clasificador elige 'erróneamente' la fuente de datos.

Enlaces de utilidad y librerías:

<https://huggingface.co/Qwen>

<https://boardgamegeek.com/boardgame/312484/lost-ruins-of-arnak/forums/0>

<https://www.nltk.org/>

<https://www.langchain.com/>

<https://www.selenium.dev/>

<https://rdflib.readthedocs.io/en/stable/>

<https://skos.um.es/TR/rdf-sparql-query/>

Ejercicio 2)

Resumen

El ejercicio 2 consta de implementar un agente que cuente con 3 herramientas: `doc_search` (para consultar en la base vectorial), `table_search` (para la base tabular) y `graph_search` (para el grafo).

Es requisito que este agente sea implementado utilizando la librería Llama Index y que esté basado en el concepto ReAct.

Desarrollo

Para el desarrollo del agente es necesario tener un modelo de lenguaje local, en el entorno local, para esto se hace uso de la librería ollama, la cual permite descargar distintos modelos open source para utilizar de forma local. El modelo implementado para el agente es llama 3.2, modelo de la empresa Meta de 3b de parámetros.

La implementación del Agente es simple, consta de definir las 3 herramientas, las cuales para este caso son los 3 retrievers desarrollados en el ejercicio 1 y crear el prompt pertinente para que el agente sepa hacer uso de las herramientas y responda con criterio.

El estilo de prompt para este caso es del estilo 'Thought', 'Action', 'Action Input'. Donde el agente piensa acerca de qué información necesita, para luego consultar la herramienta que decide con un input adecuado. Este proceso sucede iterativamente en el agente hasta poder llegar a dar una respuesta.

Ejemplos

En los siguientes casos se verán algunas situaciones en las que el agente debe recurrir a distintas a herramientas para responder:

PROMPT: "Duracion de las partidas de Las Ruinas Perdidas de Arnak"

Respuesta: 'Una partida de Las Ruinas Perdidas de Arnak dura 5 rondas.'

Prompt: "Que son los idolos?"

Respuesta: 'Los ídolos en las Ruinas Perdidas de Arnak son objetos que se pueden encontrar durante el juego y que tienen un efecto especial. Cuando se encuentran, deben estar boca abajo porque su recompensa inicial ya no se volverá a conseguir. Al final del juego, cada ídolo valdrá 3 puntos. Los jugadores pueden utilizar estos ídolos para obtener ventajas en la partida, como quedarse con un ídolo o utilizar su efecto de inmediato.'

Prompt: "Como es la preparacion para la ronda siguiente?"

Respuesta: 'Para la ronda siguiente, debes barajar todas las cartas de tu zona de juego y dejarlas en el fondo de tu mazo. Prepara el tablero para la ronda siguiente, colocando el indicador de jugador inicial en el primer jugador de la izquierda. Luego, es tu turno. Recuerda que al final de cada ronda, debes mover la vara lunar para señalar el paso del tiempo y comenzar una nueva ronda cuando crezca tu mano. La partida termina después de la quinta ronda, y en la evaluación final, cada jugador anotará los puntos correspondientes.'

Prompt: "Cuanto dura normalmente una partida?"

Respuesta: 'Una partida de Las Ruinas Perdidas de Arnak dura aproximadamente 5 rondas, durante las cuales el jugador debe explorar el templo perdido y recolectar puntos para ganar.'

Prompt: "Como es el recuento final?"

Respuesta: 'El recuento final en Las Ruinas Perdidas de Arnak se determina mediante la suma de los puntos obtenidos por cada jugador durante las 5 rondas. El jugador con más puntos al final del juego es declarado ganador.'

A continuación se comentan algunas situaciones donde el agente falla al responder.

1) Al consultar por los publicadores parece que el agente no consulto correctamente al retrieve del grafo, por lo que no obtuvo esas relaciones y respondió con parte del contexto de las reglas de juego.

Prompt: "Quiénes son los publicadores de Las Ruinas Perdidas de Arnak?"

Respuesta: "Las Ruinas Perdidas de Arnak es un juego de estrategia y aventura desarrollado por profesor kutil y su equipo. El juego se juega con 2-4 jugadores y tiene como objetivo descubrir el templo perdido en las ruinas. El juego cuenta con diferentes características y mecánicas, incluyendo la colocación de trabajadores y la construcción de estructuras."

2) En el siguiente caso responde de forma incorrecta. Esta situación se debe a que el agente tiende a responder con información que ya sabe de otros juegos de mesa.

Prompt: "Cuántos jugadores pueden jugar?"

Respuesta: "Pueden jugar entre 1 y 2 jugadores."

3) En el siguiente caso se observa una situación similar a la anterior, el agente no consulta la información correspondiente y responde con conocimiento que ya posee de otro juego.

Prompt: "Como se dispone el tablero inicialmente en Las Ruinas Perdidas de Arnak?"

Respuesta: "El tablero inicialmente se dispone con un tamaño de 63,5×88 mm, y se coloca en el centro del tablero central."

Al consultar al agente se obtienen resultados muy diversos, pudiendo llegar a responder cosas muy diferentes para el mismo prompt. Se observa que en cuestión de reglas del juego (base vectorial) suele responder de manera correcta. Sin embargo hay prompts (como por ejemplo 'Duración de una partida') que apuntan a ser respondidos mediante la información de la base tabular, el agente tiende a responder con información vectorial, y quizá no es la respuesta deseada.

Hay otras situaciones como cuando se consulta la cantidad de jugadores donde responde incorrectamente, esto puede deberse a que no obtuvo el contexto correspondiente y responde con información genérica de otros juegos, esta situación es frecuente.

Conclusiones

El agente implementado con llama3.2 presenta resultados variados, resolviendo de distintas formas incluso para una llamada con igual prompt.

Se observa que el agente inicialmente llama mal a los retrievers, intentando pasar como parámetros diccionarios u otro tipo de objetos, lo cual es incorrecto. Lo que resulta en varias llamadas a las distintas herramientas provistas. Además se observa que el agente tiende a modificar el prompt en las distintas iteraciones, lo que resulta en peores resultados llamando a los retrievers.

En líneas generales se observa que el agente termina por tomar en la mayoría de situaciones información de la base vectorial, desestimando la información tabular o el grafo.

Por otro lado hay llamadas en la que el agente responde con información general, como reglas de otros juegos de mesa, sin hacer demasiado caso al contexto que obtiene de los retrievers.

Posibles soluciones o mejoras para perfeccionar el agente:

- Utilizar prompts en inglés, los modelos open source tienden a comportarse mejor en ese idioma.
- Mejorar el prompt, destacando ejemplos e indicando llamadas a las herramientas.
- Aportar más herramientas que resuelvan acciones más puntuales y no tan generales como lo hacen los retrievers proporcionados en este trabajo.

Links de utilidad y librerías:

<https://ollama.com/>

<https://ollama.com/library/llama3.2>

<https://www.llamaindex.ai/>

https://docs.llamaindex.ai/en/stable/examples/chat_engine/chat_engine_react/