Universidad Nacional de Rosario

Facultad de Cs. Exactas, Ingeniería y Agrimensura

Procesamiento del Lenguaje Natural

Trabajo Práctico N°2

López Ceratto, Julieta : L-3311/1,

Diciembre, 2024

Figure 1: Logo Facultad Cs. Exactas, Ingeniería y Agrimensura

Introducción

En este informe se explicará de manera detallada los procedimientos realizados y resultados obtenidos para el ejercicio planteado por la cátedra. Se abordarán tanto el planteo de la solución como el código realizado y la lógica detrás del mismo.

Contents

Chapter 1

Rajas of The Ganjes

1.1 Breve explicación

Rajas of the Ganjes es un juego de mesa ambientado en la antigua India, durante el período de expansión del Imperio Mogul. Es un juego de colocación de trabajadores en el que los jugadores deberán acumular prestigio y riquezas para conseguir detonar el final de la partida y proclamarse vencedor. Los jugadores, que representan a Rajas y Ranis, están llamados a cumplir con las exigencias de su papel de venerables soberanos. Deberán mejorar sus propiedades en magnificas y ricas provincias. Teniendo en cuenta el importante papel del Karma, los jugadores equilibrarán su crecimiento en una exigente interacción de prestigio y prosperidad. El jugador con los resultados más exitosos será uno de los grandes líderes de la nación.

1.2 Objetivo

El objetivo es crear un chatbot y un agente inteligente que respondan preguntas acerca del juego basándose principalmente en información recopilada y almacenada en bases de datos tabular, vectorial y de grafos.

1.3 Abordaje

Para lograr dicho objetivo, se contemplan las siguientes acciones:

1. Elección de herramientas.
2. Armado de base de datos de grafos.
3. Armado de bases de datos vectorial.
4. Armado de bases de datos tabular.
5. Armado de consultas dinámicas y reranking para base vectorial.
6. Armado de chatbot.
7. Armado de asistente inteligente.

El ejercicio 1 del trabajo abarca los primeros 6 puntos, el ejercicio 2, el último punto y elección de herramientas específicas.

1.4 Ejercicio 1.

En este ejercicio se abarca tanto la recopilación de información para alimentar al chatbot y agente, como su respectivo almacenamiento y generación de querys dinámicas para finalmente culminar en un chatbot.

1.4.1 Punto 1: elección de herramientas.

Las herramientas utilizadas en todo el ciclo de un agente artificial y un chatbot son una rueda primordial y que determinan en gran medida su funcionamiento posterior.

Selenium

Selenium es una herramienta muy poderosa a la hora de hacer web-scrapping. Se elige esta herramienta ya que permite extraer de manera eficiente tanto información que ya se encuentra dentro del html inicial de la página, como aquella información que se carga a posteriori mediante JavaScript.

RDFlib

RDF permite estructurar los datos que alimentarán el chatbot en un formato que facilita la comprensión semántica. Esto es fundamental para que el chatbot no solo responda preguntas basadas en datos, sino que también entienda relaciones entre conceptos, proporcionando respuestas más relevantes y precisas. El uso de SPARQL como lenguaje de consulta para datos RDF permite que el chatbot acceda a información específica y estructurada en tiempo real. Esto mejora la capacidad del chatbot para responder de manera precisa incluso a consultas muy detalladas.

Chromadb

Se elige ChromaDB para el desarrollo del chatbot debido a su capacidad para almacenar y recuperar eficientemente representaciones de texto en espacios vectoriales. Esta característica permite realizar búsquedas semánticas avanzadas, mejorando la precisión de las respuestas al basarse en la similitud contextual entre las consultas del usuario y la información almacenada. A diferencia de las bases de datos tradicionales, ChromaDB está optimizada para trabajar con grandes volúmenes de datos embebidos, lo que facilita la integración con modelos de lenguaje avanzados, como los basados en arquitecturas de Transformer, y mejora significativamente la eficiencia en la recuperación de datos relevantes.

Embedding

Para realizar los embeddings que se almacenan en la base de datos, se opta por elegir un embedding multilingual, en particular: universal-sentence-encoder-multilingual. Esto se debe a que los textos están en distintos idiomas y se quiere capturar de la mejor manera posible la semántica de ellos antes de que sean traducidos.

Pandas

Para la base tabular, se elige pandas debido a su fácil implementación tanto en el armado como en las querys de consultas.

Qwen2.5-72B-Instruct

Este modelo generador de lenguaje se utiliza por su gran performance para las tareas asignadas tras varias pruebas. Se probó con otros modelos como Zephyr pero no se obtuvieron resultados al nivel de este modelo. Las tareas que abarca son: - Clasificador de una frase en temáticas dadas. - Generación de querys dinámicas para base de datos de grafo y tabular. - Chatbot para el usuario.

BM25

Esta herramienta se utiliza para realizar una búsqueda de palabras claves contra los chunks almacenados en la base de datos vectorial.

1.4.2 Armado de base de datos de grafos.

La base de datos de grafos se utiliza para modelar datos acerca del juego que no tienen que ver de manera estricta con su jugabilidad y que no son de amplia explicación, sino más bien relaciones entre distintas personas, otros juegos, premios y nacionalidades. Incluye:

- Nombres alternativos.
- Diseñadores: nombres, nacionalidad, relaciones entre ellos, premios ganados, otros juegos hechos.
- Artistas: nombre, nacionalidad, relación con diseñadores, premios, otros juegos.
- Publicadores: nombres, nacionalidad.
- Juegos de su colección / familia: nombres, relación con diseñadores y artistas del mismo juego.

Obtención de información.

Para esto, primero se extrae de la página del juego, en su sección de créditos, todos los links que redirigen a cada entidad mencionada anteriormente. Luego, se define información que se desea encontrar según cada link:

- Designer: Nombre, Pareja(si la tiene), Premios ganados, Otros juegos, nacionalidad.
- Artist: Nombre, Pareja(si la tiene), Premios ganados, Otros juegos, nacionalidad.
- Family: temática con las que se relaciona el juego.
- Publishers: Nombre.
- Nombres Alternativos: Nombre, idioma.
- Año de publicación.
- Premios: nombre de premio ganado, año.

En cada link busco la información correspondiente a cada entidad y lo almaceno en un diccionario donde las claves son las relaciones y los valores las entidades con las que tienen la relación.

Por ejemplo, para Inka Brand, creadora del juego:

```
{'Name': 'Inka Brand',
  'Partner': 'Markus Brand',
  'Award': '2012 Spiel des Jahres Kennerspiel des Jahres
Winner',
  'Other Games': ['Andor Junior: Die Fährtenleserin Fennah / Der
Fährtenleser Fenn (2020)',
   'Andor: The Family Fantasy Game (2020)',
   'Andor: The Family Fantasy Game – The Danger in the Shadows
(2022)',
   'Bibi & Tina: Das Spiel zum Film (2013)',
   'Bitte nicht öffnen: Bissig! (2022)',
   'Black Stories: Das Spiel (2022)',
   'Blitzed! (2013)',
   'La Boca (2013)',
   'Bravo! (2019)',
   'Burgenland (2014)'],
  'Nacionalidad': 'German'},
```

Figure 1.1: Diccionario obtenido para Inka Brand.

Armado de base de grafos.

La base de grafos que se arma es la siguiente:

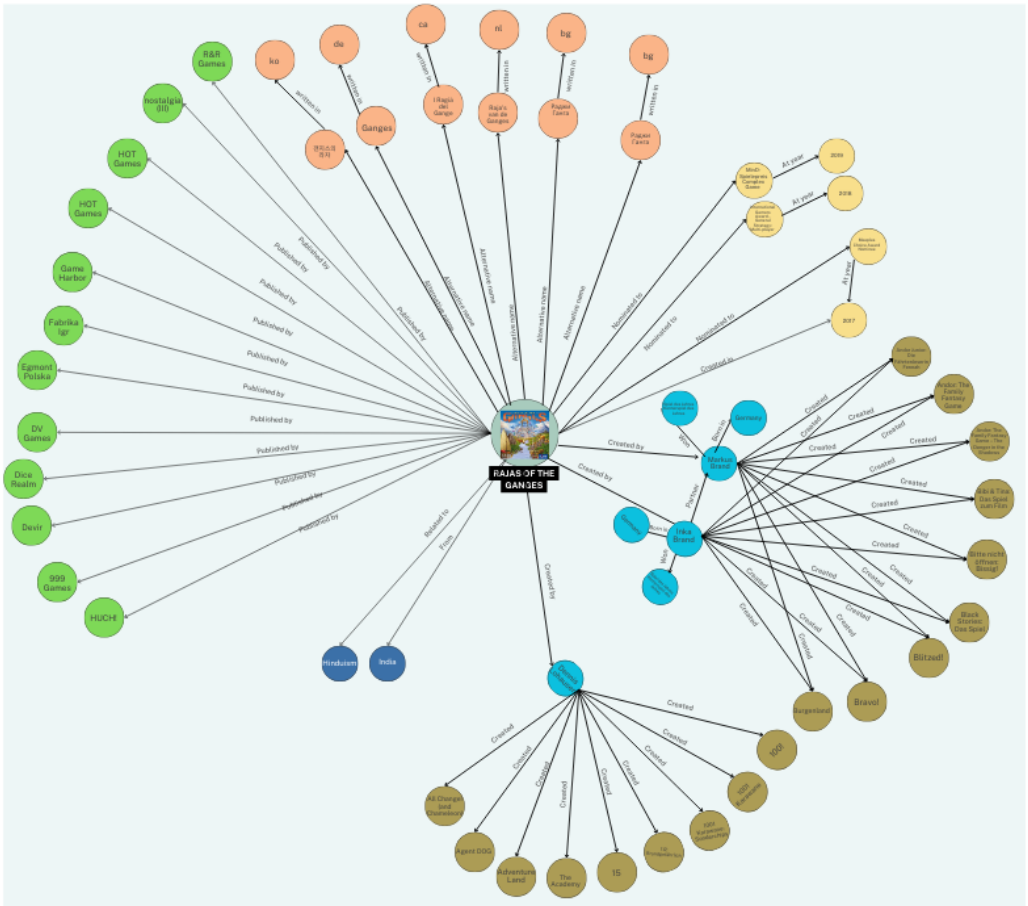


Figure 1.2: Figura base de grafos

Se agrega la entidad del juego 'Rajas of the Ganjes', la cual está relacionada de manera directa o indirecta con todas las entidades y su información obtenida. Se agregan los nodos con sus aristas, obteniendo la base de grafos ilustrada anteriormente. Finalmente, si se desea consultar, por ejemplo, el año de publicación: Se obtiene el año 2017.

```
query = """
    SELECT (STRAFTER(STR(?Year), "http://wiki_rajas.org/") AS ?
name)
    WHERE {
        ?game a <http://wiki_rajas.org/Game> .
        ?game <http://wiki_rajas.org/Release_year> ?Year .
    }
"""
 # Ejecutar consulta y mostrar resultados
results = g.query(query)
for row in results:
    print(row.name)
```

Figure 1.3: Consulta manual base de grafos.

1.4.3 Armado de base de datos vectorial.

La base de datos vectorial tendrá el fin de almacenar:

- Reglas del juego generales.
- Información sobre las casillas.
- Información sobre el objetivo, juego en general.
- Opiniones.

Extracción de información

Para la información de las reglas, tenemos 3 fuentes de datos:

1. El documento en español del manual original del juego.
2. PDF de reglas resumidas tanto para el modo normal como nirvana.

Información de casillas

1. El documento en español del manual original del juego.

Información sobre el objetivo, juego en general

1. El documento en español del manual original del juego.

Opiniones

1. Foro de opiniones de usuarios.
2. Video.

- El documento en español del manual original del juego está dividido en secciones, por lo que se opta por extraer el texto en secciones ya que luego sirve para la meta-data a la hora de añadir la información a la base vectorial.

  Las secciones son: 'OBJETIVO DEL JUEGO', 'SETUP','TRANSCURSO DEL JUEGO', 'ACCIÓN CONSTRUIR (CANTERA)', 'TABLERO DE PROVINCIA', 'Las mejoras de edificios', 'ACCIÓN DE MERCADO (MERCADO)', 'ACCIÓN DE PALACIO', 'ACCIÓN DE RÍO (Puerto)', 'Karma', 'Obtener nuevos trabajadores.', 'Bonos en los tracks de puntuación.', 'FIN DE RONDA', 'FINAL DE PARTIDA', 'NOTAS', 'VERSIÓN NAVARATNAS', 'EL MÓDULO GANGA', 'MOGUL LEXIGON', 'CRÉDITOS'.

  De esta forma, se obtiene un diccionario 'sections' que almacena la información en las secciones mencionadas.

- En segundo lugar, se tienen dos manuales cortos: uno de guía rápida y otro de referencias rápidas, ambos con reglas resumidas y concisas del juego y algunos otros detalles.

- En tercer lugar, se tiene un documento más asociado a las reglas del modo autómata del juego.

- En cuarto lugar, se extrae información de foros; estos están relacionados a: ¿Cuándo el jugador Portuguese es útil?, Estrategias para principiantes, estrategias para modo competitivo y reviews ("Rajas of the Ganges: An Awesome Game that Exceeded My Expectations", "Silver Duck Reviews: Rajas of the Ganges", "A Disappointing First Impression of Rajas", "Patrick Reviews Rajas of the Ganges")

- Como último lugar, se extrae información de un video review del juego.

Limpieza y transformación de texto.

Una vez obtenida esa información, se realiza una limpieza de texto en primera medida, tanto para eliminar caracteres de html con el fin de pasar el texto a una forma más clara, eliminando tildes que puedan ser confusas para un modelo del lenguaje, entre otros.

Luego se divide en chunks los textos. Para esto, se utiliza RecursiveSplitter con un tamaño de 400 y un overlap de 30. Esta elección se hizo mediante prueba y error. A un tamaño muy grande, resultaba confuso para el modelo, uno muy pequeño, no llegaba a abarcar la información deseada.

Los chunks quedan almacenados en conjunto con su categoría. Por ejemplo, los chunks correspondientes a 'OBJETIVO DEL JUEGO':

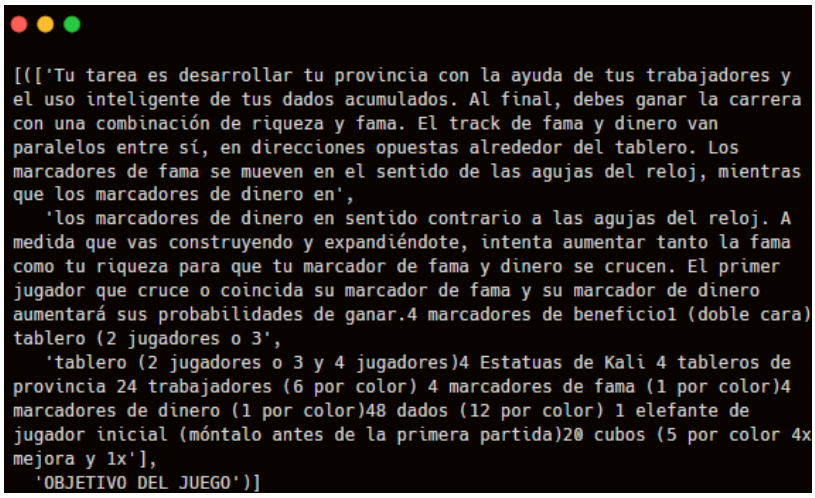


Figure 1.4: Chunks objetivos del juego.

Armado de base vectorial y carga de información.

Se arma una base vectorial chromadb; para la carga de los chunks a la base vectorial, se realiza una función que hace lo siguiente:

1. Embedding del chunk.
2. Metadata del chunk: 'Metadata recibida': 'metada específicada en el chunk'
3. ID: f"id recibido+numero de chunk"
4. Carga a la base vectorial.

Se llama a la función de la siguiente manera para los chunks separados:

```
fill_db(chunk_reglas_generales, {'Reglas del juego': 0}, 'RG')
fill_db(chunk_reglas_especificas, {'Reglas del juego':0}, 'RE')
fill_db(chunk_estrategias, {'Estrategias': 0}, 'EST')
fill_db(chunk_opinones, {'Opinion': 'opinion del juego'}, 'OP')
fill_db(chunk_objetivo_juego, {'Objetivo del juego': 0}, 'OBJ')
```

Figure 1.5: Llamada a función fill db.

Y se van cargando los chunks de la siguiente manera:

```
ID: RG_0, Embedding Length: 512, Metadata: {'Reglas del juego': 'SETUP'},
texto: El jugador que haya sacado el valor más bajo en los dados se convierte
en el jugador principal. En caso de empate, el último jugador que comió
comida india. El jugador inicial colocará el elefante junto a su estatua de
Kali y pondrá su marcador de riqueza en el espacio 3 en el track de riqueza.
El jugador a su izquierda 4, el siguiente 5, y así sucesivamente. El jugador
que vaya último comenzará
ID: RG_1, Embedding Length: 512, Metadata: {'Reglas del juego': 'TRANSCURSO
DEL JUEGO'}, texto: El Juego se desarrolla en varias rondas,...
```

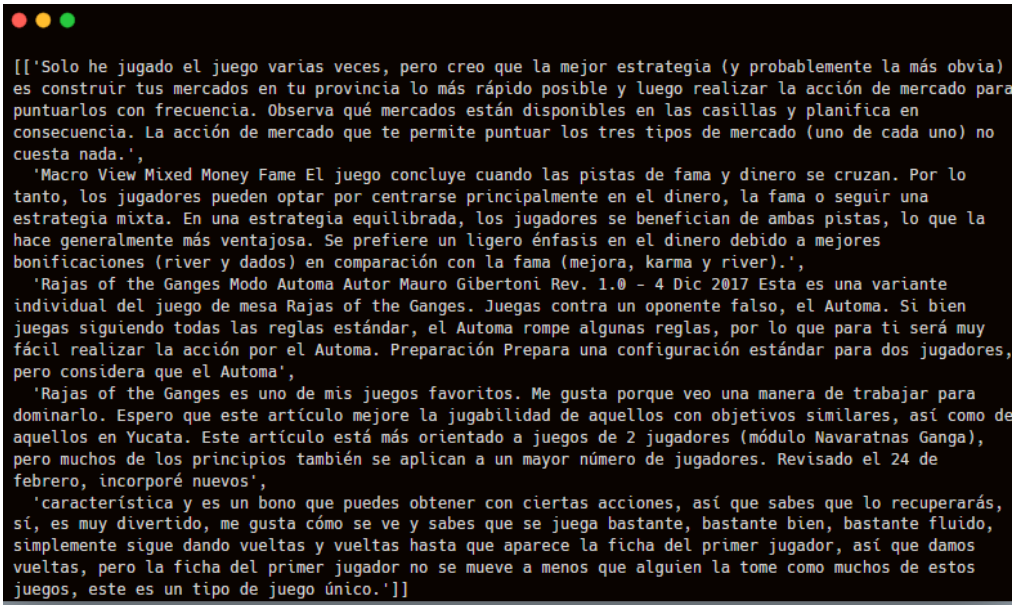
Figure 1.6: Ejemplo fill db.

Finalmente, se realiza una llamada de ejemplo:

```
consulta = "Estrategia para principiantes"
embedding_consulta = embed([consulta]).numpy().tolist()
results = vec_db.query(
    query_embeddings=embedding_consulta,
    n_results=5,

)
```

Figure 1.7: Llamada de ejemplo a base vectorial.

Y se obtienen los siguientes documentos:



```
[['Solo he jugado el juego varias veces, pero creo que la mejor estrategia (y probablemente la más obvia)
es construir tus mercados en tu provincia lo más rápido posible y luego realizar la acción de mercado para
puntuarlos con frecuencia. Observa qué mercados están disponibles en las casillas y planifica en
consecuencia. La acción de mercado que te permite puntuar los tres tipos de mercado (uno de cada uno) no
cuesta nada.',
  'Macro View Mixed Money Fame El juego concluye cuando las pistas de fama y dinero se cruzan. Por lo
tanto, los jugadores pueden optar por centrarse principalmente en el dinero, la fama o seguir una
estrategia mixta. En una estrategia equilibrada, los jugadores se benefician de ambas pistas, lo que la
hace generalmente más ventajosa. Se prefiere un ligero énfasis en el dinero debido a mejores
bonificaciones (river y dados) en comparación con la fama (mejora, karma y river).',
  'Rajas of the Ganges Modo Automa Autor Mauro Gibertoni Rev. 1.0 - 4 Dic 2017 Esta es una variante
individual del juego de mesa Rajas of the Ganges. Juegas contra un oponente falso, el Automa. Si bien
juegas siguiendo todas las reglas estándar, el Automa rompe algunas reglas, por lo que para ti será muy
fácil realizar la acción por el Automa. Preparación Prepara una configuración estándar para dos jugadores,
pero considera que el Automa',
  'Rajas of the Ganges es uno de mis juegos favoritos. Me gusta porque veo una manera de trabajar para
dominarlo. Espero que este artículo mejore la jugabilidad de aquellos con objetivos similares, así como de
aquellos en Yucata. Este artículo está más orientado a juegos de 2 jugadores (módulo Navaratnas Ganga),
pero muchos de los principios también se aplican a un mayor número de jugadores. Revisado el 24 de
febrero, incorporé nuevos',
  'característica y es un bono que puedes obtener con ciertas acciones, así que sabes que lo recuperarás,
sí, es muy divertido, me gusta cómo se ve y sabes que se juega bastante, bastante bien, bastante fluido,
simplemente sigue dando vueltas y vueltas hasta que aparece la ficha del primer jugador, así que damos
vueltas, pero la ficha del primer jugador no se mueve a menos que alguien la tome como muchos de estos
juegos, este es un tipo de juego único.']]
```

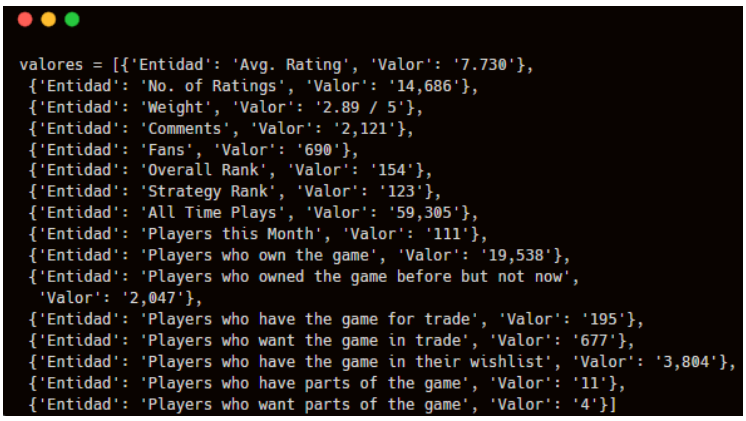
Figure 1.8: Documentos obtenidos tras llamada de prueba

1.4.4 Armado de base tabular.

La base tabular tendrá como objetivo almacenar:

- Datos estadísticos: jugadores totales, por mes, etc.
- Dificultad del juego.
- Cantidad de fichas.
- Precio promedio entre sitios web.
- Tiempo de juego
- Edad recomendada.
- Puntuación.

Se realiza un web-scraping sobre el apartado 'stats' de la página del juego y se obtiene un diccionario como el siguiente:



```
valores = [{'Entidad': 'Avg. Rating', 'Valor': '7.730'},
 {'Entidad': 'No. of Ratings', 'Valor': '14,686'},
 {'Entidad': 'Weight', 'Valor': '2.89 / 5'},
 {'Entidad': 'Comments', 'Valor': '2,121'},
 {'Entidad': 'Fans', 'Valor': '690'},
 {'Entidad': 'Overall Rank', 'Valor': '154'},
 {'Entidad': 'Strategy Rank', 'Valor': '123'},
 {'Entidad': 'All Time Plays', 'Valor': '59,305'},
 {'Entidad': 'Players this Month', 'Valor': '111'},
 {'Entidad': 'Players who own the game', 'Valor': '19,538'},
 {'Entidad': 'Players who owned the game before but not now',
  'Valor': '2,047'},
 {'Entidad': 'Players who have the game for trade', 'Valor': '195'},
 {'Entidad': 'Players who want the game in trade', 'Valor': '677'},
 {'Entidad': 'Players who have the game in their wishlist', 'Valor': '3,804'},
 {'Entidad': 'Players who have parts of the game', 'Valor': '11'},
 {'Entidad': 'Players who want parts of the game', 'Valor': '4'}]
```

Figure 1.9: Diccionario base tabular.

Luego, simplemente se lo pasa a DataFrame de Pandas obteniendo así la base tabular:

	Entidad	Valor
0	Avg. Rating	7.730
1	No. of Ratings	14,688
2	Weight	2.89 / 5
3	Comments	2,121
4	Fans	690
5	Overall Rank	154
6	Strategy Rank	123
7	All Time Plays	59,331
8	Players this Month	133
9	Players who own the game	19,540
10	Players who owned the game before but not now	2,049
11	Players who have the game for trade	194
12	Players who want the game in trade	676
13	Players who have the game in their wishlist	3,805
14	Players who have parts of the game	11
15	Players who want parts of the game	4

Figure 1.10: Enter Caption

1.4.5 Armado de clasificadores.

Se deben armar clasificadores a fin de, a raíz de una frase, obtener la base de datos en la que se debe buscar.

Para esto, se vale de las categorías:

"ESTADISTICA GENERAL", 'DISEÑADORES', 'CREADORES', 'IDIOMAS DEL JUEGO', 'PUBLICADORES DEL JUEGO', 'REGLAS GENERALES', 'REGLAS ESPECÍFICAS AUTOMATA', 'REGLAS ESPECIFICAS NAVARATNA', 'OBJETIVO DEL JUEGO', 'ESTRATEGIA COMPETIDORES', 'ESTRATEGIA PRINCIPIANTES', 'ESTRATEGUIA PERSONAJE PORTUGUESE', 'AÑO DE CREACIÓN', 'OTROS IDIOMAS'

Se intenta con dos clasificadores: uno de regresión logística y otro mediante un modelo Qwen 2.5. El clasificador de regresión logística no tiene una performance sumamente mala, sin embargo, tiene problemas para diferenciar creadores de diseñadores, y con algunas otras categorías también.

```
[159] clasificador_consulta.predict(embed(['Quien fue el creador del juego?']))
      array(['INFORMACION CREADORES DEL JUEGO'], dtype=object)

[160] clasificador_consulta.predict(embed(['quien creo el juego?']))
      array(['ESTADÍSTICA GENERAL DEL JUEGO'], dtype=object)

[161] clasificador_consulta.predict(embed(['Cual es el objetivo del juego?']))
      array(['OBJETIVO DEL JUEGO'], dtype=object)

[162] clasificador_consulta.predict(embed(['Qué puntaje tiene el juego?']))
      array(['ESTADÍSTICA GENERAL DEL JUEGO'], dtype=object)
```

Figure 1.11: Ejemplo clasificador regresion logística.

Por otro lado, el modelo Qwen 2.5, diferencia de manera más clara las categorías:

```
[139] get_class('Cual es el objetivo del juego?')
      'OBJETIVO DEL JUEGO'

[140] get_class('Cuándo gana un jugador?')
      'OBJETIVO DEL JUEGO'

[141] get_class('De que forma comienza el jugador a jugar?')
      'REGLAS GENERALES'

[142] get_class('¿que puedo hacer para ganar?')
      'ESTRATEGIA PRINCIPIANTES'

[143] get_class('¿Para que puedo usar al personaje portuguese?')
      'ESTRATEGIA PERSONAJE PORTUGUESE'

[144] get_class('Como hago para jugar el juego?')
      'REGLAS GENERALES'

[145] get_class('quienes crearon el juego?')
      'CREADORES'
```

Figure 1.12: Ejemplo clasificador qwen

Finalmente, se hace una función que, recibida una categoría, devuelve la base de datos a utilizar.

```python
def get_database(clas):

    tematica_vector_db = ['REGLAS GENERALES', 'REGLAS ESPECÍFICAS AUTOMATA',
'REGLAS ESPECIFICAS NAVARATNA', 'OBJETIVO DEL JUEGO','ESTRATEGIA COMPETIDORES',
'ESTRATEGIA PRINCIPIANTES', 'ESTRATEGUIA PERSONAJE PORTUGUESE']
    tematica_graph_db = ['DISEÑADORES', 'CREADORES', 'IDIOMAS DEL JUEGO',
'PUBLICADORES DEL JUEGO', 'AÑO DE CREACIÓN', 'OTROS IDIOMAS']
    tematica_tabular_db = ["ESTADISTICA GENERAL"]

    if clas in tematica_vector_db:
        return 'vector_db'
    if clas in tematica_graph_db:
        return 'graph_db'
    if clas in tematica_tabular_db
        return 'tabular_db'
```

Figure 1.13: Función clasificadora de base de datos.

1.4.6 Armado de querys dinámicas.

Se realiza un modelo para la base tabular y otro para la base de grafos con Qwen 2.5 cuyo objetivo sea armar querys de consulta para dichas bases.

Query dinámica para base de grafos.

Al modelo se le especifica el siguiente prompt:

```python
chat_prompt = [{
        "role": "system",
        "content": f"""
        Eres un modelo que realiza consultas a una base de datos RDF sobre el juego de mesa "Rajas
of the Ganjes" a raíz de una frase del usuario.
        La base tiene la siguiente estructura: las entidades son {entidades}, las relaciones son
{relaciones}.
        Asegúrate de diferenciar bien entre diseñadores y creadores.
        Debes escribir SOLAMENTE UNA consulta URI con SPARQL que se adecue a la frase del usuario.
        Es importante que solo respondas con una sola consulta.
        Por ejemplo, si recibes la frase: '¿Cuándo se creó el juego?' deberías solamente
        devolver: "SELECT (STRAFTER(STR(?Year), 'http://wiki_rajas.org/') AS ?name) WHERE {{ ?game
a <http://wiki_rajas.org/Game> . ?game <http://wiki_rajas.org/Release_year> ?Year }}"
        Es importante que respetes todos los corchetes, llaves y entidades luego de ?."""
    }, {
        "role": "user",
        "content": input_text
    }]
```

Figure 1.14: Prompt modelo query graph

Donde entidades:

```
[
    "Rajas_of_the_Ganges", "Inka_Brand", "Markus_Brand", "Dennis_Lohausen",
    "HUCH!", "999_Games", "Devir", "Dice_Realm", "DV_Games", "Egmont_Polska",
    "Fabrika_Igr", "Game_Harbor", "HOT_Games", "nostalgia (III)", "R&R_Games",
    "India", "Hinduism", "Ganges", "I Ragià del Gange", "Raja's van de Ganges",
    " ", "", " ", "2019 - MinD-Spielepreis Complex Game Nominee",
    "2018 - Juego del Año Recommended", "2018 - International Gamers Award \
    - General Strategy: Multi-player Winner",
    "2018 - International Gamers Award
```

```
    -General Strategy: Multi-player Nominee",
    "2017 - Meeples Choice Award Nominee", "2017"
]
```

Y relaciones:

```
    [
    ("Rajas_of_the_Ganges", "designedBy", ["Inka_Brand", "Markus_Brand"]),
    ("Rajas_of_the_Ganges", "artBy", ["Dennis_Lohausen"]),
    ("Rajas_of_the_Ganges", "publishedBy", [
        "HUCH!", "999_Games", "Devir", "Dice_Realm", "DV_Games", "Egmont_Polska",
        "Fabrika_Igr", "Game_Harbor", "HOT_Games", "nostalgia (III)", "R&R_Games"
    ]),
    ("Rajas_of_the_Ganges", "setIn", ["India"]),
    ("Rajas_of_the_Ganges", "religionAssociatedWith", ["Hinduism"]),
    ("Rajas_of_the_Ganges", "alsoKnownAs", [
        "Ganges", "I Ragià del Gange", "Raja's van de Ganges", " ",
        "", " "
    ]),
    ("Rajas_of_the_Ganges", "awardedOrNominated", [
        "2019 - MinD-Spielepreis Complex Game Nominee",
        "2018 - Juego del Año Recommended",
        "2018 - International Gamers Award - General Strategy: Multi-player Winner",
        "2018 - International Gamers Award - General Strategy: Multi-player Nominee",
        "2017 - Meeples Choice Award Nominee"
    ]),
    ("Rajas_of_the_Ganges", "Release_year", ["2017"])
]
```

El modelo genera de forma correcta las consultas. Por ejemplo, si se le pide la consulta para la pregunta "Quién publicó el juego?" devuelve:

```
SELECT (STRAFTER(STR(?Publisher), '\href{http://wiki_rajas.org/}
{http://wiki\_rajas.org/}') AS ?name)
WHERE \{ ?game a <\href{http://wiki_rajas.org/Game}
{http://wiki\_rajas.org/Game}> .
?game <\href{http://wiki_rajas.org/publishedBy}
{http://wiki\_rajas.org/publishedBy}> ?Publisher \}
```

Query dinámica base tabular.

Se le pasa el siguiente prompt al modelo:

```
chat_prompt = [{
        "role": "system",
        "content": f"""
        Eres un modelo que realiza consultas a una base de datos tabular en dataframe de pandas
sobre el juego de mesa "Rajas of the Ganjes" a raíz de una frase del usuario.
        La base tiene la siguiente estructura: las columnas son: {df_valores.columns}, donde los
valores son:
        Entidad: {df_valores['Entidad']}.
        Avg.rating contiene el puntaje medio del juego.
        Debes escribir SOLAMENTE UNA consulta de búsqueda en dataframe que se adecue a la frase del
usuario.
        Es importante que solo respondas con una sola consulta.
        Por ejemplo, si recibes la frase: '¿Cuántos jugadores han jugado el juego?' deberías
solamente
        devolver: 'df_valores[df_valores['Entidad'] == 'All Time Plays']['Valor'].values[0]'"""
    }, {
        "role": "user",
        "content": input_text
    }]
```

Figure 1.15: Prompt modelo query tabular.

Donde las entidades son: Avg. Rating', 'No. of Ratings', 'Weight', 'Comments', 'Fans', 'Overall Rank', 'Strategy Rank', 'All Time Plays', 'Players this Month', 'Players who own the game', 'Players who owned the game before but not now', 'Players who have the game for trade', 'Players who want the game in trade', 'Players who have the game in their wishlist', 'Players who have parts of the game', 'Players who want parts of the game'

Y las columnas de la base: 'Entidad, Valor'

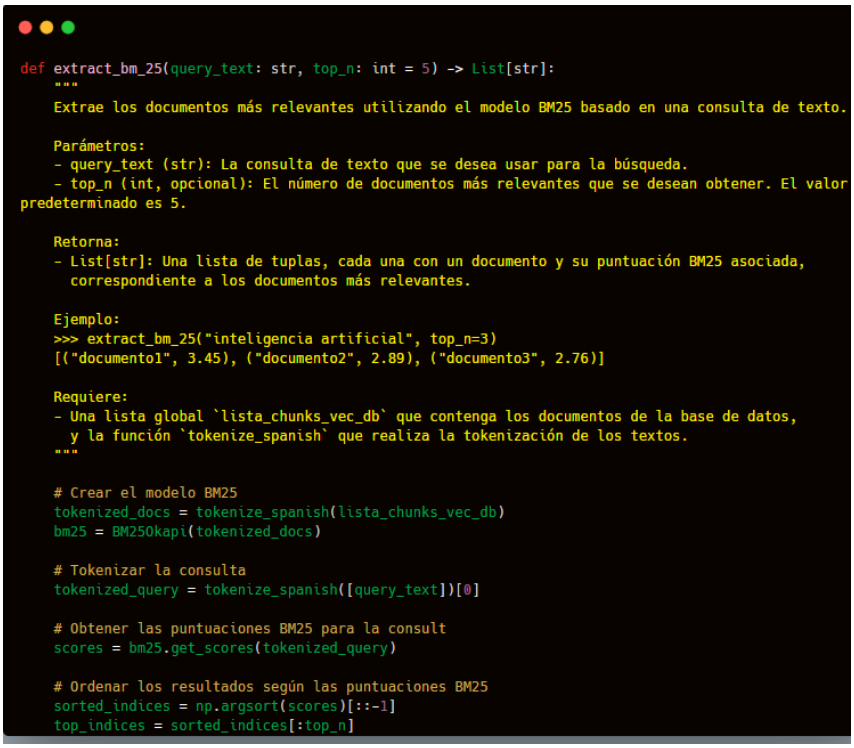
Por ejemplo, si se le solicita al modelo la query para la pregunta "¿Cuántos jugadores han jugado el juego?", devuelve: "df_valores[df_valores['Entidad'] == 'All Time Plays']['Valor'].values[0] "

Consulta dinámica para base vectorial.

La consulta dinámica se compone de 3 funciones. La primera, hace una búsqueda por palabra clave utilizando bm25; la segunda busca por comparación semántica a través del método .query() de la base de datos vectorial.

Finalmente, la tercera compara los resultados de las dos búsquedas y rerankea los resultados, devolviendo los primeros 5 mejores.

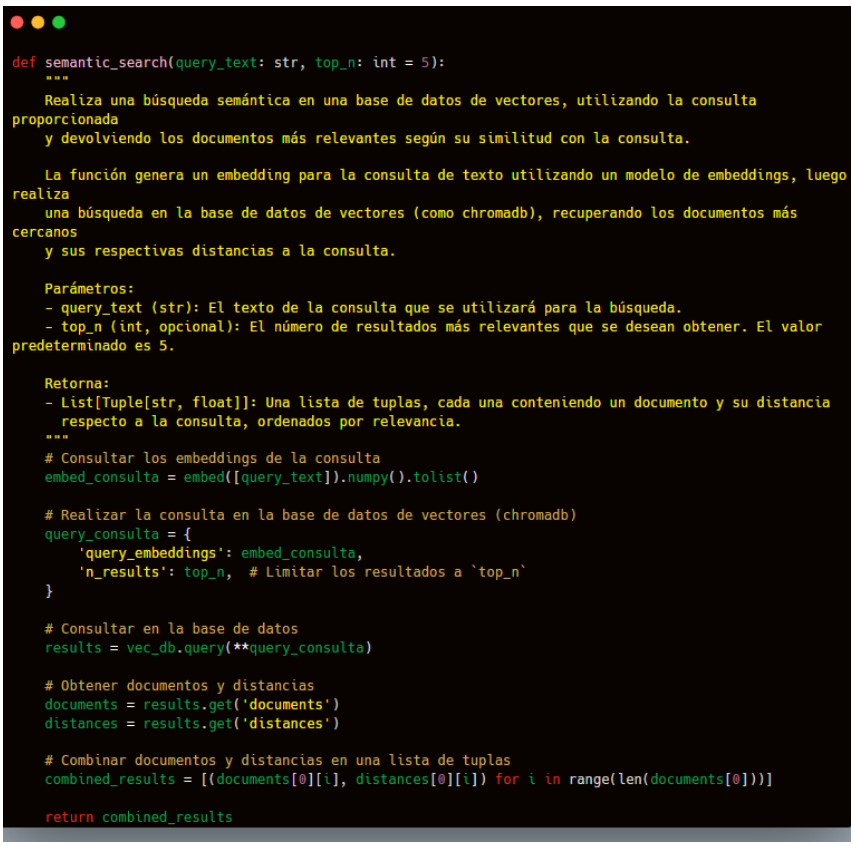
En cuanto a la primera función, lo que hace es utilizar todos los chunks de la base vectorial, tokenizarlos y a partir de ahí recibe la consulta del usuario, la tokeniza y hace una búsqueda por palabra clave contra los chunks tokenizados. Devuelve los documentos encontrados con sus respectivos scores.



```python
def extract_bm_25(query_text: str, top_n: int = 5) -> List[str]:
    """
    Extrae los documentos más relevantes utilizando el modelo BM25 basado en una consulta de texto.

    Parámetros:
    - query_text (str): La consulta de texto que se desea usar para la búsqueda.
    - top_n (int, opcional): El número de documentos más relevantes que se desean obtener. El valor
predeterminado es 5.

    Retorna:
    - List[str]: Una lista de tuplas, cada una con un documento y su puntuación BM25 asociada,
      correspondiente a los documentos más relevantes.

    Ejemplo:
    >>> extract_bm_25("inteligencia artificial", top_n=3)
    [("documento1", 3.45), ("documento2", 2.89), ("documento3", 2.76)]

    Requiere:
    - Una lista global `lista_chunks_vec_db` que contenga los documentos de la base de datos,
      y la función `tokenize_spanish` que realiza la tokenización de los textos.
    """

    # Crear el modelo BM25
    tokenized_docs = tokenize_spanish(lista_chunks_vec_db)
    bm25 = BM25Okapi(tokenized_docs)

    # Tokenizar la consulta
    tokenized_query = tokenize_spanish([query_text])[0]

    # Obtener las puntuaciones BM25 para la consult
    scores = bm25.get_scores(tokenized_query)

    # Ordenar los resultados según las puntuaciones BM25
    sorted_indices = np.argsort(scores)[::-1]
    top_indices = sorted_indices[:top_n]
```

Figure 1.16: Bm25 función.

La segunda, hace el embedding de la consulta y luego pasa ese embedding al método.query() en el formato adecuado. Finalmente, devuelve los documentos encontrados con la distancia de cada uno hacia la consulta.



```python
def semantic_search(query_text: str, top_n: int = 5):
    """
    Realiza una búsqueda semántica en una base de datos de vectores, utilizando la consulta
proporcionada
    y devolviendo los documentos más relevantes según su similitud con la consulta.

    La función genera un embedding para la consulta de texto utilizando un modelo de embeddings, luego
realiza
    una búsqueda en la base de datos de vectores (como chromadb), recuperando los documentos más
cercanos
    y sus respectivas distancias a la consulta.

    Parámetros:
    - query_text (str): El texto de la consulta que se utilizará para la búsqueda.
    - top_n (int, opcional): El número de resultados más relevantes que se desean obtener. El valor
predeterminado es 5.

    Retorna:
    - List[Tuple[str, float]]: Una lista de tuplas, cada una conteniendo un documento y su distancia
      respecto a la consulta, ordenados por relevancia.
    """
    # Consultar los embeddings de la consulta
    embed_consulta = embed([query_text]).numpy().tolist()

    # Realizar la consulta en la base de datos de vectores (chromadb)
    query_consulta = {
        'query_embeddings': embed_consulta,
        'n_results': top_n,  # Limitar los resultados a `top_n`
    }

    # Consultar en la base de datos
    results = vec_db.query(**query_consulta)

    # Obtener documentos y distancias
    documents = results.get('documents')
    distances = results.get('distances')

    # Combinar documentos y distancias en una lista de tuplas
    combined_results = [(documents[0][i], distances[0][i]) for i in range(len(documents[0]))]

    return combined_results
```

Figure 1.17: Búsqueda semántica funció.

La tercera, se encarga de rerankear tanto lo encontrado en la búsqueda semántica como por palabras clave. Para esto, convierte los scores de palabras clave para que se encuentren dentro del mismo rango que las distancias de la búsqueda semántica. Luego los ordena y obtiene los mejores 5, devolviéndolos.

```python
def rerank_results(query_text: str, top_n: int = 5) -> List[str]:
    """
    Realiza un re-ranking entre los resultados obtenidos por BM25 y la búsqueda semántica.
    Escala los puntajes de BM25 al rango 0-2 para que coincidan con las distancias de ChromaDB.

    Args:
        query_text (str): La consulta de búsqueda.
        top_n (int): Número de resultados principales a devolver (por defecto 5).

    Returns:
        List[str]: Los documentos re-rankeados.
    """
    # Obtener los resultados de BM25
    bm25_results = extract_bm_25(query_text, top_n=top_n)

    # Obtener los resultados de la búsqueda semántica (ChromaDB)
    semantic_results = semantic_search(query_text, top_n=top_n)

    # Obtener los puntajes de BM25 para normalizarlos al rango 0-2
    bm25_scores = [bm25_score for _, bm25_score in bm25_results]
    min_bm25_score = min(bm25_scores)
    max_bm25_score = max(bm25_scores)

    # Crear un diccionario para combinar los puntajes
    combined_results = {}

    # Añadir los resultados de BM25 al diccionario, escalados al rango 0-2
    for doc, bm25_score in bm25_results:
        # Normalización Min-Max al rango 0-2
        if max_bm25_score != min_bm25_score:
            scaled_bm25_score = 2 * (bm25_score - min_bm25_score) / (max_bm25_score - min_bm25_score)
        else:
            scaled_bm25_score = 1  # Si todos los puntajes de BM25 son iguales, asignar un valor arbitrario (1 en este caso)

        combined_results[doc] = {'bm25': scaled_bm25_score, 'semantic': 0}

    # Añadir los resultados de la búsqueda semántica al diccionario
    for doc, semantic_distance in semantic_results:
        semantic_score = 2 - (semantic_distance)  # Convertir distancia a similitud (0 -> 2, 2 -> 0)
        if doc not in combined_results:
            combined_results[doc] = {'bm25': 0, 'semantic': semantic_score}
        else:
            combined_results[doc]['semantic'] = semantic_score

    # Crear una lista con los documentos y sus puntajes combinados
    reranked_docs = [
        (doc, combined_results[doc]['bm25'] + combined_results[doc]['semantic'])
        for doc in combined_results
    ]

    # Ordenar los resultados por el puntaje combinado
    reranked_docs_sorted = sorted(reranked_docs, key=lambda x: x[1], reverse=True)

    # Devolver los documentos re-rankeados
    return reranked_docs_sorted[:top_n]
```

Figure 1.18: Rerank función.

Por ejemplo, para la pregunta "¿Qué jugador comienza el juego?" se obtienen los siguientes documentos rerankeados:

```
[('Excepto en el caso en el que quieras apresurarte para llevar a tu trabajador al puente, no puedo entender por qué es útil usar un 6 en los
portugueses. Cuanto antes atravieses el río, menos probabilidades tendrás de beneficiarte de esas bonificaciones de espacio de 1 río y reclamar
recompensas causadas por otros eventos. ¿De qué otra manera podrían ser útiles los portugueses?',
  2.0),
 ('El jugador que haya sacado el valor más bajo en los dados se convierte en el jugador principal. En caso de empate, el último jugador que comió
comida india. El jugador inicial colocará el elefante junto a su estatua de Kali y pondrá su marcador de riqueza en el espacio 3 en el track de
riqueza. El jugador a su izquierda 4, el siguiente 5, y así sucesivamente. El jugador que vaya último comenzará',
  1.5027467564597794),
 ('Tan pronto como un jugador logre que su marcador de fama y su marcador de riqueza se equiparen en valores o se crucen, pasándose el uno al
otro, se dispara el final de la partida. La ronda actual se terminará. Es decir, todos los jugadores que estén entre el jugador que ha activado el
final de la partida y el jugador inicial actual todavía tendrán la oportunidad de ganar y podrá colocar un',
  0.7028152942657471),
 ('característica y es un bono que puedes obtener con ciertas acciones, así que sabes que lo recuperarás, sí, es muy divertido, me gusta cómo se
ve y sabes que se juega bastante, bastante bien, bastante fluido, simplemente sigue dando vueltas y vueltas hasta que aparece la ficha del primer
jugador, así que damos vueltas, pero la ficha del primer jugador no se mueve a menos que alguien la tome como muchos de estos juegos, este es un
tipo de juego único.',
  0.6685147285461426),
 ('GANGES (2 a 4 jugadores) Antes de empezar Decidir si se juega en modo normal o en modo Navaratna (ver el epígrafe Variante Navaratna más
adelante ). Sortear jugador inicial , el cual se va rotando en sentido horario cada vez que se agoten los lacayos colocados sobre las acciones del
tablero, a menos que un jugador obtenga el beneficio de la primera zona de los aposentos del palacio, que lo',
  0.6675455570220947)]
```

Figure 1.19: Salida ejemplo rerank

Se puede observar que, a pesar de escalar los scores de bm25, si este es muy alto, de todas maneras obtiene prioridad.

1.4.7 Armado de chatbot para usuario.

El chatbot se arma con Qwen 2.5, al igual que los modelos vistos anteriormente. Se realiza una clase que contiene:

```python
class llm_rajas_of_the_ganjes():
    """
    Esta clase define un chatbot interactivo diseñado para responder consultas sobre el juego *Rajas of The Ganges*
    utilizando un modelo de lenguaje y diferentes fuentes de datos.

    La clase permite realizar consultas, obtener contextos de diversas bases de datos (como bases de datos vectoriales,
    gráficas o tabulares), y generar un prompt para interactuar con un modelo de lenguaje.

    Métodos:
    - __init__: Inicializa el cliente de inferencia y define los atributos necesarios para el chatbot.
    - get_context: Obtiene el contexto necesario para responder a una consulta, basándose en el tipo de base de datos.
    - make_prompt: Crea un prompt estructurado para la interacción con el modelo de lenguaje.
    - hablar: Inicia una conversación con el chatbot, permite al usuario hacer consultas y recibir respuestas del modelo.
    """

    def __init__(self):
        """
        Inicializa el cliente de inferencia (InferenceClient) con una clave de API y establece los atributos
        iniciales para el chatbot.
        """
        self.client_rotg = InferenceClient(api_key=get_token())
        self.info = None
        self.chat_prompt = None

    def get_context(self, consulta):
        """
        Obtiene el contexto adecuado para una consulta dada, determinando el tipo de base de datos (vectorial, gráfica o tabular).

        Dependiendo de la base de datos asignada, la función consulta y recupera los datos necesarios
        para proporcionar una respuesta relevante.

        Parámetros:
        - consulta (str): La consulta del usuario para la que se necesita obtener el contexto.

        Retorna:
        - self.info (List or None): La información contextual relevante obtenida de la base de datos.
        """
        classification = get_class(consulta)
        db = get_database(classification)
        if db == 'vector_db':
            self.info = rerank_results(consulta)
        elif db == 'graph_db':
            query = get_query(consulta)
            self.info = [result.name[0:] for result in g.query(query)]
        elif db == 'tabular_db':
            query = get_tabular_query(consulta)
            self.info = [eval(query)]
        else:
            self.info = None

        return self.info

    def make_prompt(self, consulta):
        """
        Crea un prompt estructurado para interactuar con el modelo de lenguaje, utilizando el contexto
        obtenido de la base de datos y la consulta del usuario.

        Parámetros:
        - consulta (str): La consulta del usuario para la que se necesita generar el prompt.

        Retorna:
        - chat_prompt (List[Dict]): Una lista de diccionarios que contiene el prompt estructurado con
          el rol de sistema y usuario para la interacción con el modelo.
        """
        chat_prompt = [{
            "role": "system",
            "content": f"""
                Eres un modelo que responde preguntas acerca del juego *Rajas of The Ganges*.
                Es importante que sólo respondas en base a la siguiente información: {self.info}.
                Es muy importante que contestes en oración. Si la información proporcionada es un número, responde con ese número como parte de la
oración.
                Sólo si la información es "None", responde 'No tengo información para responder la consulta'."""
        }, {
            "role": "user",
            "content": consulta
        }]
        return chat_prompt

    def hablar(self):
        """
        Inicia una conversación interactiva con el usuario, permitiendo realizar consultas y recibir respuestas
        generadas por el modelo de lenguaje. El ciclo de conversación se mantiene hasta que el usuario escribe
        'chau' para salir.

        El ciclo de interacción involucra la obtención de contexto, la creación del prompt y la generación de respuestas
        del modelo, que se imprimen como respuestas del agente.
        """
        consulta = 'c'
        print('Bienvenido a Rajas of The Ganjes Chatbot! para salir escribe "chau"')
        while True:
            consulta = input("Escribe tu consulta, para salir despídete con 'chau'':")
            if consulta == 'Chau' or consulta == 'chau'
              print('Gracias por usar Rajas of The Ganjes Chatbot, hasta la próxima!')
              break
            print(f"User: {consulta}")
            self.info = self.get_context(consulta)
            self.chat_prompt = self.make_prompt(consulta)
            # Realizar la solicitud de completado de chat usando el cliente
            completion = self.client_rotg.chat.completions.create(
                model="Qwen/Qwen2.5-72B-Instruct",
                messages=self.chat_prompt,
                max_tokens=500,
                temperature=0.5
            )
            print(f"Agente: {completion.choices[0].message['content']}")
```

Figure 1.20: Clase chatbot.

Donde tiene una función para obtener el contexto que luego utilizará en el prompt. Esto lo hace a raíz de una consulta ingresada, obteniendo su clasificación temática y luego la base de datos donde buscar; como tercer paso, utiliza el llm adecuado o búsqueda reranking para consultar a la base de datos y obtener así el contexto a utilizar en el prompt.

Se obtiene la siguiente interacción con el chatbot:

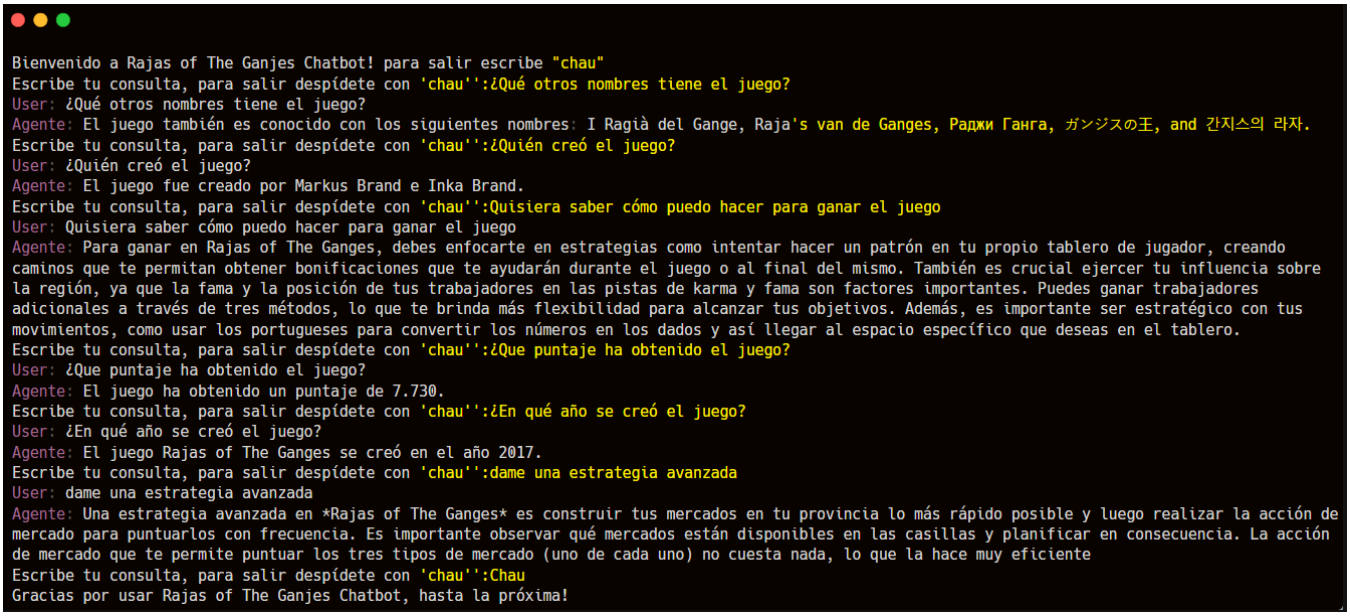


Figure 1.21: Interacción con modelo.

1.5 Ejercicio 2: Agente Inteligente.

1.5.1 Herrmientas específicas elegidas.

Se eligió el modelo Llama3.2:3b ya que fue el que mejor resultado se obtuvo frente a las consultas del usuario y respecto a otros modelos. Los otros modelos probados fueron phi:3 y llama3.2:1b. Se descargó el modelo a través de Ollama.

1.5.2 Herramientas provistas al modelo.

Las herramientas o tools que se le proveen al modelo son las fabricadas anteriormente. Ellas son:

- `get_info_graph_db()`: obtiene la query de la consulta y con dicha query consulta a la base de grafos. Retorna la respuesta a la consulta a la base.

- `get_info_tabular_db()`: obtiene la query de la consulta y con dicha query consulta a la base tabular. Retorna la respuesta a la consulta a la base.

- `get_info_vector_db()`: obtiene los mejores documentos rerankeados de la base vectorial a raíz de una consulta. Retorna dichos documentos.

Observación: se decidió no incorporar la herramienta de búsqueda de Wikipedia ya que el modelo tenía una tendencia a utilizarla de forma compulsiva y responder con dicha información-

1.5.3 Armado del agente.

Parámetros

Al agente se le da una temperatura muy baja de 0.1; esto se hace ya que se busca que no divague mucho en las respuestas y se centre en la información de contexto que tiene y recupera a raíz de sus herramientas.

El context window es de 4096, un tamaño acorde a las finalidades del modelo, donde se tiene que brindar respuestas de no mucha extensión y donde se recibe información corta para responderlas.

Además se cerciora de que en dicha ventana de contexto entren los documentos rerankeados de las consultas donde se utilice la búsqueda en la base vectorial de forma tal que no se quede información por fuera.

Prompt.

```
    system_prompt="""Estás diseñado para responder preguntas acerca del juego 'Rajas of the Ganges' utilizando **exclusivamente
información de las bases de datos tabular, vectorial y de grafos, brindada por las herramientas que se mencionana a
continuación***. Asegúrate de seguir estrictamente las instrucciones para cada consulta.

      ## Herramientas disponibles:
      1. **get_info_graph_db**: Busca información en la base de datos de grafos. Parámetro: texto de consulta.
      2. **get_info_tabular_db**: Busca información en la base de datos tabular. Parámetro: texto de consulta.
      3. **get_info_vector_db**: Busca información en la base de datos vectorial. Parámetro: texto de consulta.

      ### Temáticas de cada base:
      - **Grafos**: DISEÑADORES, CREADORES, IDIOMAS DEL JUEGO, PUBLICADORES DEL JUEGO, AÑO DE CREACIÓN DEL JUEGO, OTROS IDIOMAS.
      - **Tabular**: ESTADÍSTICA GENERAL.
      - **Vectorial**: REGLAS GENERALES, REGLAS ESPECÍFICAS AUTOMATA, REGLAS ESPECÍFICAS NAVARATNA, OBJETIVO DEL JUEGO, ESTRATEGIA
COMPETIDORES, ESTRATEGIA PRINCIPIANTES, ESTRATEGIA PERSONAJE PORTUGUESE.

      ### Instrucciones para cada consulta:
      1. **Analiza la consulta** para determinar la información necesaria.
      2. **Usa las herramientas disponibles una por una**, pasando como parámetro **exactamente** la consulta recibida.
      3. **Combina los resultados** en una respuesta final clara y concisa.

      ### Reglas adicionales:
      - **Nunca modifiques la consulta del usuario**: Cada vez que recibas una consulta nueva, **olvida la anterior**. La nueva
consulta debe ser tratada de manera independiente y sin influencias previas.
      - **Prioriza las herramientas** según la relación temática con la consulta. Asegúrate de seleccionar la herramienta que mejor
se ajuste a la consulta de acuerdo con las temáticas proporcionadas.
      - Si la pregunta requiere información de **más de una herramienta**, usa **todas** las herramientas necesarias para obtener
una respuesta completa.
      - **Nunca** respondas con información que no provenga directamente de las herramientas.
      - El formato de salida debe ser **estricto** y seguir el ejemplo que se proporciona.

      ### Ejemplo de interacción:
      #### Consulta 1:
      "¿Cuándo finaliza la partida?"
      - **Pensamiento (Thought)**: Necesito buscar información sobre las reglas del juego.
      - **Acción (Action)**: `get_info_vector_db`
      - **Entrada de acción (Action Input)**: "¿Cuándo finaliza la partida?"
      - **Observación (Observation)**: "La partida finaliza cuando un jugador logra que sus marcadores de fama y riqueza se
equiparen..."
      - **Respuesta final (Final Answer)**: La partida finaliza cuando un jugador logra que sus marcadores de fama y riqueza se
equiparen, lo que dispara el final de la partida.

      #### Consulta 2:      "Año de creación del juego?"
      - **Pensamiento (Thought)**: Necesito buscar información sobre el año de creación del juego.
      - **Acción (Action)**: `get_info_graph_db`
      - **Entrada de acción (Action Input)**: "Año de creación del juego"
      - **Observación (Observation)**: "El juego fue creado en 2017."
      - **Respuesta final (Final Answer)**: El juego fue creado en 2017.

      ### IMPORTANTE:
      - **No uses información de consultas anteriores** para la nueva consulta
      - **Nunca alteres la consulta recibida**. Siempre debes utilizar el texto exacto como parámetro.
      - **Siempre asegúrate de usar todas las herramientas necesarias** antes de no dar una respuesta completa.
      """
      ,
            react_chat_history=False,
          context="""
          Eres un asistente útil que responde en español. Recuerda las salidas de todas las herramientas para usarlas en la
respuesta final.
          """)
```

Figure 1.22: Prompt agente inteligente.

Luego, para interactuar con el usuario, se define la siguiente función:

```python
def chat_con_agente(query: str):
    """
    Función para interactuar con el agente ReAct.
    """
    try:
        if not query.strip():
            return "La consulta está vacía"

        response = agent.chat(query)
        return response
    except Exception as e:
        return f"Error al procesar la consulta:
{str(e)}"
```

Figure 1.23: Función para interactuar con el usuario del agente inteligente.

Resultados con preguntas de prueba.

Los resultados obtenidos con las siguientes preguntas de prueba fueron:

- "Necesito una estrategia para principiantes " -¿ La estrategia para principiantes en Rajas of the Ganges es construir mercados en tu provincia lo más rápido posible y luego realizar la acción de mercado para puntuarlos con frecuencia. También es importante observar qué mercados están disponibles en las casillas y planificar en consecuencia. Un ligero énfasis en el dinero puede ser beneficioso debido a mejores bonificaciones en comparación con la fama.

  Para conseguir esta respuesta, primero consultó en la base de grafos y tabular, donde no consiguió información; luego en la vectorial, donde cosigue información e identifica que está asociada a la pregunta. Finalmente responde.

- "Año de creación del juego?" -¿El juego Rajas of the Ganges fue lanzado en el año 2017. Para armar esta respuesta, consigue información de la base vectorial, como la información de la base de grafos era confusa (su pensamiento es: "I was unable to extract the año de creación del juego from the output."), continuó a la de grafos donde obtuvo el año de creación de manera clara. Finalmente responde.

- "¿Quién feu el creador del juego?" -¿ Los creadores del juego Rajas of the Ganges son Inka Brand y Markus Brand. Inicia con un pensamiento inicial, probablemente de información pre-aprendida. "El juego Rajas of the Ganges fue creado por Asmodee." la cual es incorrecta. Acierta en buscar en la base de grafos pero pasa mal el parámetro por lo que obtiene un error; de allí pasa a utilizar la búsqueda en la base vectorial, donde obtiene información pero no es del todo convincente (su pensamiento, de hecho, es: La herramienta proporcionó información sobre el juego Rajas of the Ganges, incluyendo su creador y otros detalles. Sin embargo, la información no fue lo que esperaba.). Vuelve a intentar buscar en la de grafos donde finalmente, al pasar de forma correcta el parámetro, consigue la información correcta. Arma la respuesta y la presenta al usuario.

- Con qué otros nombres se conoce el juego? -¿ El juego Rajas of the Ganges también se conoce como Rajas de las Ganges en español, pero no tengo información adicional sobre otros nombres. No logra encontrar la información, si bien busca de forma adecuada en la base de grafos, hay un error en la generación de la query probablemente porque modifica la consulta al pasársela. Finalmente responde.

- "De qué nacionalidad son sus creadores?" -¿El juego Rajas of the Ganges fue creado por Inka Brand y Markus Brand, quienes son de nacionalidad italiana. Si bien logra recordar quienes eran los creadores, consigue de manera errónea su nacionalidad. Esto se debe a que busca en una base de datos errónea y extrae un texto que no es el adecuado para la respuesta, sumando a que no logra distinguir que no es informaciónr relacionada a los creadores. Además de que en los textos proporcionados por la base vectorial no menciona ni al país Italia ni derivados de la palabra, lo que sugiere que armó la respuesta en base a conocimientos previos. Finalmente responde.

- "Con qué religión se asocia el juego? "-¿ El juego Rajas of the Ganges se relaciona con la religión hindú y es un juego de colocación de trabajadores que simula la competencia por ganar riqueza y fama en el contexto del río Ganges. Obtiene la respuesta acertada. Aunque no busca la información en la base de grafos, que es la que tiene la temática asociada a la pregunta, sino que busca en la vectorial. En la de grafos la busca como segunda opción de herramienta, pero modifica la consulta pasada por lo que no obtiene información de dicha base. Finalmente responde.

- "Qué puntaje obtuvo el juego? " -¿El puntaje del juego Rajas of the Ganges es de 7,73. Obtiene la respuesta correcta. Primero busca en la base de grafos, lo cual es incorrecto, de allí no consigue respuesta. Luego busca en la tabular, lo que le proporciona la información adecuada. Finalmente responde.

- ¿Cuántos jugadores en el mes tuvo? -¿Error al procesar la consulta: Reached max iterations. No pudo obtener la respuesta porque se excedió en iteraciones. Por un lado, cabe destacar que sí consultó a la base adecuada de acorde a la temática de la pregunta, sin embargo tiene errores al parsear el output, es decir, no pudo armar el formato de salida pensamiento - accion - input-observación, lo que hizo que no pueda continuar con la búsqueda y armado de respuesta.

Inconvenientes y planteo de mejoras.

El agente tiene dos problemas principales: Por un lado, tiene un gran inconveniente con mantener la consulta del usuario para pasarla tal como es en el parámetro de sus herramientas, lo que provoca errores u no obtención de información. Por otro lado, la identificación correcta de la base a utilizar, a pesar de las indicaciones de que se rija por el uso en orden de mejor a peor herramienta en términos de coincidencia con la temática de la consulta.

Algunas mejoras pueden ser:

- Implementar una validación dentro del proceso de toma de decisiones para evitar cualquier tipo de modificación de la consulta antes de pasarla como parámetro.

- Implementar un sistema de ponderación o relevancia para seleccionar la herramienta más adecuada según la temática de la consulta. Esto puede lograrse asignando un puntaje a cada herramienta en función de la relación con la consulta, mejorando la precisión en la selección.

1.6 Conclusión.

El armado tanto de un chatbot como de un agente inteligente es una tarea que debe ser muy cuidadosa en todos sus pasos. La fuente de información es una parte clave: el buscar información clara, de calidad y confiable hará que la efectividad del modelo sea favorecida. Por otro lado, se puede ver las distintas finalidades y usos para las diferentes formas de almacenamiento de la información:

- Una base de datos de grafos es sumamente útil a la hora de modelar relaciones clave entre distintas entidades, permitiéndole al modelo obtener información clara acerca de esto que no se vea de alguna manera afectada por la redacción de las relaciones en formado de oraciones o párrafos.

- Una base de datos tabular es muy útil para almacenar valores numéricos, estadísticos o consisos que no necesariamente tienen una relación entre sí, pero sí con la temática general del modelo. Similar a lo que pasa con la de grafos, aquí el modelo obtiene la respuesta justa a la consulta asociada a dicha información, permitieno de esta manera una identificación adecuada del dato a devolver.

- Una base de datos vectorial: es muy potente a la hora de almacenar información que debe ser interpretada en un contexto y con una explicación linguística más avanzada que la información almacenada en las bases anteriores. Se ve muy influenciada tanto por el tamaño de chunks, como el embedding elegido y la metadata proporcionada. Permite una búsqueda semántica muy fructuosa para el modelo a la hora de tener que responder a consultas amplias como pueden ser "Quiero una estrategia para principiantes" o bien "¿Qué jugador arranca la partida?" Además, permite no sólo una búsqueda demántica sino también por palabras clave (si bien no se implementó dentro de la base vectorial sino con bm25 y sus chunks).

Otra cuestión contemplada fue la importancia de conciliar las búsquedas tanto por palabras clave como por similitud semántica. Una búsqueda sólo por palabras clave puede ser engañosa, no siempre un párrafo que contenga la o las palabras claves de la consulta del cliente será apropiado para responder. Tal es el caso de "¿Qué jugador arranca la partida?" Una oración que sea "El jugador arranca la pieza del tablero cuando pierde en una ronda de la partida", técnicamente contiene todas las palabras claves, pero semánticamente no es acorde. Por otro lado, una búsqueda sólo semántica, si bien puede dar resultado, a veces puede dejar de lado frases que podrían ser relevantes a la hora de armar una respuesta a la consulta del usuario; siguiendo con el ejemplo anterior, frases como "El turno inicial está determinado por las reglas oficiales del juego" o "El jugador designado como primero siempre inicia la acción." tienen relación conceptual con la pregunta y podrían ser útiles. Sin embargo, una oración como "El jugador que empieza suele ser elegido al azar mediante un sorteo," aunque relevante, podría no ser seleccionada si la relación semántica no se detecta correctamente. Es por esto que combinar ambas búsquedas mediante un rerankin se hace muy importante ya que de esta manera se obtienen las ventajas de ambas búsquedas, haciendo pasar a segundo plano sus desventajas.

En cuanto a experiencia con ambos modelos (chatbot y agente inteligente), se encontró en este caso una mejor con el primero, ya que de una manera rápida proveía respuestas acordes a la pregunta; sin embargo, depende fuertemente de cómo esté programada la búsqueda, y se sustenta en modelos clasificadores por detrás para esto. Por lo que se podría decir que es un modelo basado en conjuntos de otros modelos.

Por otro lado, el agente inteligente se observó que era más autónomo; de manera automática buscaba la herramienta a utilizar y podía hacer inferencias a raíz de la información encontrada, así como armar respuestas más "humanas" a la percepción del usuario. Aunque bien, esto a costa de

algunos errores durante su procesamiento de la consulta y de una consulta a varias herramientas hasta encontrar la respuesta más adecuada. Otro punto a destacar y en relación a esto último es que, al no estar limitado a una herramienta, pudo obtener respuestas más completas utilizando información de las otras fuentes de datos donde el chatbot nunca podría haber obtenido la información por su forma de estar programado.

En fin, tanto el chatbot como el agente tienen sus puntos fuertes y débiles.

1.7 Bibliografía.

- BoardGameGeek. (n.d.). *Rajas of the Ganges: Credits*. BoardGameGeek. https://boardgamegeek.com/boardgame/220877/rajas-of-the-ganges/credits [16-12-2024]
- Python Software Foundation. (n.d.). *Python*. Python.org. https://www.python.org/https://www.python.org/ [16-12-2024]
- Hugging Face. (n.d.). *Qwen2.5-1.5B-Instruct*. Hugging Face. https://huggingface.co/Qwen/Qwen2.5-1.5B-Instructhttps://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct [16-12-2024]
- Ollama. (n.d.). *Llama3.2*. Ollama. https://ollama.com/library/llama3.2https://ollama.com/library/llama3.2 [16-12-2024]
- Chroma. (n.d.). *Chroma*. Chroma. https://www.trychroma.com/https://www.trychroma.com/ [16-12-2024]
- Python Software Foundation. (n.d.). *Langdetect*. PyPI. https://pypi.org/project/langdetect/https://pypi.org/project/langdetect/ [16-12-2024]
- Google. (n.d.). *TensorFlow*. TensorFlow. https://www.tensorflow.org/?hl=es-419https://www.tensorflow.org/?hl=es-419 [16-12-2024]
- Ollama. (n.d.). *Ollama*. Ollama. https://ollama.com/https://ollama.com/[16-12-2024]
- Hugging Face. (n.d.). *Hugging Face*. Hugging Face. https://huggingface.co/https://huggingface.co/ [16-12-2024]
- Python Software Foundation. (n.d.). *Pydub*. PyPI. https://pypi.org/project/pydub/https://pypi.org/project/pydub/ [16-12-2024]
- Python Software Foundation. (n.d.). *SpeechRecognition*. PyPI. https://pypi.org/project/SpeechRecognition/https://pypi.org/project/SpeechRecognition/ [16-12-2024]
- Python Software Foundation. (n.d.). *PyPDF2*. PyPI. https://pypi.org/project/PyPDF2/https://pypi.org/project/PyPDF2/ [16-12-2024]

- Nidhaloff. (n.d.). *Deep-translator*. GitHub. MIT License.

  https://github.com/nidhaloff/deep-translatorhttps://github.com/nidhaloff/deep-translator [16-12-2024]

- spaCy. *spaCy*. spaCy.

  https://spacy.io/https://spacy.io/ [16-12-2024]

- spaCy. *es_core_news_sm*. spaCy.

  https://spacy.io/models/es#es$_c ore_n ews_s mhttps : //spacy.io/models/es#es_core_news_sm[16-12-2024]PythonSoftwareFoundation.(n.d.).llama-index-llms-ollama.PyPI.

  https://pypi.org/project/llama-index-llms-ollama/https://pypi.org/project/llama-index-llms-ollama/ [16-12-2024]

- LitelLM. (n.d.). *Simple proxy*. LitelLM.

  https://docs.litellm.ai/docs/simple$_p roxyhttps$: //docs.litellm.ai/docs/simple_proxy[16-12-2024]LlamaIndex.(n.d.).LlamaIndex.LlamaIndex.

  https://www.llamaindex.ai/https://www.llamaindex.ai/ [16-12-2024]

- Python Software Foundation. (n.d.). *rank-bm25*. PyPI.

  https://pypi.org/project/rank-bm25/https://pypi.org/project/rank-bm25/ [16-12-2024]

- NLTK Project. (n.d.). *NLTK: Natural Language Toolkit*. NLTK.

  https://www.nltk.org/https://www.nltk.org/ [16-12-2024]

- Selenium Project. (n.d.). *Selenium*. Selenium.

  https://www.selenium.dev/https://www.selenium.dev/ [16-12-2024]