
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

# Procesamiento del Lenguaje Natural
# Trabajo Práctico No. 2

17/12/2024

Docentes: Juan Pablo Manson, Alan Geary
Giuliano Crenna (C-7438/1)

# Contenidos

# 1 Introducción

El presente informe constituye el Trabajo Práctico Final de la materia TUIA NLP 2024, centrado en el diseño y desarrollo de un chatbot experto basado en la técnica de *Retrieval Augmented Generation* (RAG) aplicado al juego de mesa *Las ruinas perdidas de Arnak*. Este trabajo tiene como objetivo integrar conocimientos adquiridos a lo largo de la cursada, implementando soluciones avanzadas de procesamiento de lenguaje natural, recuperación de información y generación de texto.

El proyecto se divide en dos ejercicios principales. En el primero, se implementa un sistema capaz de interactuar con los usuarios, respondiendo preguntas en español e inglés mediante consultas dinámicas a diversas fuentes de datos, como documentos textuales, tablas y bases de datos de grafos. En el segundo, se extiende esta funcionalidad al incorporar un agente basado en el modelo ReAct, que utiliza herramientas específicas para buscar información en dichas fuentes, maximizando la eficacia y precisión de las respuestas.

La temática propuesta por la cátedra, *Las ruinas perdidas de Arnak*, es un Eurogame destacado, lo que añade un desafío adicional al tratamiento de los datos disponibles y a la generación de respuestas contextualmente relevantes. Este informe detalla las decisiones de diseño, implementación y justificación de cada componente desarrollado, presentando además un análisis crítico de los resultados obtenidos y propuestas de mejora.

# 2 Ejercicio 1

## 2.1 Resumen

El código presentado aborda un problema de procesamiento y análisis de información relacionada con el juego de mesa "Las Ruinas Perdidas de Arnak". Este problema incluye la descarga y preprocesamiento de datos en diferentes formatos (texto, audio, PDF y CSV), la construcción de un modelo de clasificación basado en Random Forest, y la creación de un sistema de recuperación de información empleando bases de datos vectoriales y modelos de lenguaje.

## 2.2 Desarrollo

### Fuentes de datos

Se tomaron las siguientes fuentes de datos para la posterior creación de las bases de datos.

1. Reseñas de paginas web.
2. Manual en PDF.
3. Transcripciones de videos de YouTube.

### 2.2.1 Preparación de Datos

1. Se descargan los archivos necesarios desde fuentes externas utilizando comandos `wget`, incluyendo reseñas, reglas del juego en PDF, audio y un archivo CSV de preguntas y respuestas. 2. Se implementan funciones específicas para procesar diferentes tipos de archivos:

- Extracción de texto desde páginas web y archivos de texto plano.
- Extracción de texto desde documentos PDF usando `PdfReader`.
- Procesamiento de archivos de audio en formato M4A, dividiendo en fragmentos y utilizando `speech_recognition` para la transcripción.
- Lectura y transformación de datos CSV, preparándolos para su inserción en una base de datos vectorial.

3. Los textos procesados se dividen en fragmentos manejables usando el `RecursiveCharacterTextSplit` lo que permite una mejor manipulación y almacenamiento.

### 2.2.2 Creación de la Base de Datos Vectorial

Se utiliza `chromadb` para gestionar una base de datos vectorial persistente. Los datos procesados son indexados empleando embeddings generados con el modelo `all-MiniLM-L6-v2` de `SentenceTransformer`.

### 2.2.3 Recuperación y Expansión de Información

1. Se implementa un sistema de recuperación de información que permite realizar consultas a la base vectorial. 2. Las consultas se enriquecen con información relevante recuperada de la base vectorial para generar prompts más completos. 3. La generación de respuestas se realiza mediante la API de OpenAI, incorporando el contexto ampliado.

### 2.2.4 Construcción del RAG

El desarrollo del agente RAG se compone de los siguientes pasos clave:

- **Inicialización del Agente:** Se define una clase `Agente` que incluye los elementos necesarios: un modelo de embeddings (`SentenceTransformer`), una base de datos vectorial (*ChromaDB*), y un archivo CSV para consultas tabulares.
- **Consulta Tabular:** Mediante la función `consultar_csv()`, el sistema busca coincidencias exactas o parciales en el archivo CSV. Si encuentra una coincidencia para la pregunta, devuelve la respuesta asociada.
- **Búsqueda Semántica:** Si no hay respuesta en el CSV, se utiliza la función `realizar_busqueda_semantica()`, que emplea embeddings generados por el modelo para buscar respuestas relevantes en la base de datos vectorial.
- **Predicción de Respuesta:** La función `predecir_respuesta()` combina ambas fuentes de información: primero consulta el CSV, luego recurre a la base vectorial si es necesario. Además, actualiza el contexto del agente con la respuesta seleccionada.

- **Interacción con el Usuario:** El método `chat()` permite una interacción continua con el usuario. Las respuestas generadas se basan en las consultas realizadas, utilizando el contexto acumulado para enriquecer las interacciones.

El modelo se apoya en un flujo híbrido, asegurando que las respuestas sean precisas y contextualmente relevantes, aprovechando tanto datos estructurados como no estructurados.

### 2.2.5 Construcción del Modelo de Clasificación

1. Se utiliza un modelo `Random Forest` para clasificar preguntas basándose en sus características extraídas mediante `TfidfVectorizer`. 2. El dataset de preguntas y respuestas se transforma, etiquetando las respuestas de forma categórica. 3. El conjunto de datos se divide en entrenamiento y prueba. El modelo es entrenado con `RandomForestClassifier` y evaluado usando un reporte de clasificación.

## 2.3 Conclusiones

El código logra integrar múltiples fuentes de datos y formatos en un flujo coherente, combinando técnicas de preprocesamiento, recuperación de información y clasificación. La implementación del modelo de Random Forest proporciona una herramienta eficiente para predecir respuestas basadas en preguntas de usuarios, mientras que la base vectorial y los modelos de lenguaje enriquecen la experiencia de búsqueda de información.

Sin embargo, se identificaron puntos de mejora como:

- Optimización del procesamiento de grandes volúmenes de datos en la base vectorial.
- Ampliación del conjunto de datos para cubrir más preguntas y respuestas.
- Refinamiento del modelo de clasificación para mejorar la precisión.

## 2.4 Enlaces

- Repositorio de datos: `https://github.com/giulicrenna/TUIA-NLP`
- Documentación de ChromaDB: `https://www.chromadb.org`
- Documentación de SentenceTransformers: `https://www.sbert.net`
- Documentación de OpenAI: `https://platform.openai.com/docs`

# 3 Ejercicio 2

## 3.1 Resumen

En este ejercicio, se analiza un agente implementado en Python que combina capacidades de búsqueda semántica y consultas tabulares para responder preguntas. Este agente utiliza un modelo de embeddings, una base de datos vectorial (ChromaDB) y un archivo CSV con preguntas y respuestas predefinidas. A continuación, se describen los pasos principales de la implementación, su desarrollo y las soluciones adoptadas.

## 3.2 Desarrollo

### 3.2.1 Inicialización del agente

El agente se inicializa con los siguientes componentes:

- **Modelo de embeddings:** Encargado de codificar las preguntas en vectores.
- **Base de datos vectorial:** ChromaDB, utilizada para búsquedas semánticas.
- **Archivo CSV:** Contiene preguntas y respuestas predefinidas para consultas tabulares.

El constructor de la clase también permite incluir un contexto opcional que se actualizará durante el uso del agente. Los datos del CSV se cargan utilizando la librería pandas.

### 3.2.2 Módulos principales

1. **Actualizar contexto:** Método que permite agregar nueva información al contexto del agente.
2. **Consulta sobre el CSV:** Este método busca coincidencias con la pregunta en la columna correspondiente del archivo CSV y devuelve la respuesta asociada si existe. Utiliza coincidencias parciales e ignora diferencias de mayúsculas y minúsculas.
3. **Búsqueda semántica:** Realiza una consulta en la base de datos vectorial. El modelo de embeddings convierte la pregunta en un vector y se buscan las respuestas más cercanas en el espacio vectorial. Por defecto, devuelve las 5 respuestas más relevantes.
4. **Predicción de respuesta:** Este método combina las funcionalidades de búsqueda en el CSV y la base de datos vectorial. Si una respuesta se encuentra en el CSV, se devuelve directamente. Si no, se realiza la búsqueda semántica y se selecciona la respuesta más relevante. Además, el contexto del agente se actualiza con la respuesta final.
5. **Chat interactivo:** Permite mantener una conversación continua con el agente. Las preguntas y respuestas se registran en el contexto, y se puede salir del chat escribiendo comandos como "salir" o "adiós".

### 3.2.3 Implementación del modelo de embeddings y configuración

Se utiliza el modelo `sentence-transformers/paraphrase-MiniLM-L12-v2` para codificar las preguntas y respuestas. Este modelo está diseñado para tareas de semántica textual y proporciona representaciones compactas y eficientes.

## 3.3 Conclusiones

El agente presentado es una herramienta versátil que combina dos métodos complementarios para responder preguntas: consultas tabulares y búsqueda semántica. La inclusión de un contexto dinámico mejora su capacidad para adaptarse a conversaciones prolongadas. Sin embargo, algunas limitaciones incluyen: - Dependencia de la calidad y amplitud de las respuestas en el CSV y la base de datos vectorial. - Dificultades para manejar preguntas ambiguas o fuera del dominio entrenado.

El sistema podría mejorarse incorporando: - Fuentes adicionales de datos, como API externas. - Métodos de evaluación de confianza para las respuestas generadas.

## 3.4 Enlaces

ción de ChromaDB (https://docs.trychroma.com/)

gs en Hugging Face (https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L12-v2)

Librería pandas (https://pandas.pydata.org/)