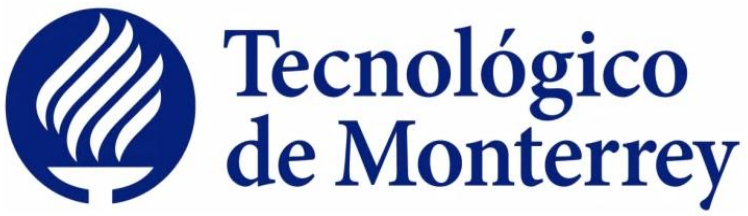

TC5035.10 Proyecto Integrador

Avance 2. Ingeniería de características

Jesús Ariel Cortés Sarmiento A01552580

Grupo:
45

05 de octubre del 2025
Monterrey, N.L

Contenido

Embedding de las transcripciones

En esta fase del proyecto, se realizó el embedding de los textos que habían sido transcritos en la fase previa. Empezamos con el archivo all_transcriptions.txt que se había generado usando Whisper, dentro de este archivo cada sesión estaba identificada con un "--- {nombre de la sesión} ---" seguida por el texto correspondiente a la grabación. El objetivo ahora era transformar las transcripciones a embeddings.

El primer paso fue dividir el texto en chunks, de modo que cada chunk fuera un fragmento coherente. Primero el texto fue dividido en secciones en donde cada una era una sesión diferente, cada sesión fue identificada con el separador "--- {nombre de la sesión} ---" que estaba al inicio de cada transcripción, con el fin de que no hubiera oraciones de dos sesiones distintas en el mismo chunk, y que cada chunk se pudiera identificar a cada sesión cuando se haga el RAG. Posteriormente, cada sesión se dividió en oraciones, cada una identificada al encontrar un símbolo especial. Finalmente, se creo un chunk al que se asigno la primera oración, y se contabilizó a cuantos tokens equivalía usando la librería tiktoken, si al agregar la siguiente oración la cantidad total de tokens era menor a los 400 tokens, se agregaba al mismo chunk, de lo contrario se asignaba a uno nuevo. Se escogieron 400 tokens tras observar de manera cualitativa con distintos limites en una pequeña muestra de texto en que punto se mantenían ideas lo suficientemente robustas sin llegar a un contexto demasiado amplio. Con este primer paso se crearon 1,872 chunks en total.

Como segundo paso, se utilizó el text-embedding-3-small para realizar el embedding de cada uno de los chunks. Se decidió utilizar este embedder debido a que cuenta con una buena calidad a un costo muy bajo y alta velocidad. Se tomo en cuenta la posibilidad de que la creación de los chunks pueda necesitar ajustes una vez se llegue a la fase de evaluaciones, por lo que de ser necesario volver a realizar el proceso de embedding, los recursos perdidos serían 6 veces menores de si se hubiera usado el modelo grande (OpenAI, 2025).

Finalmente, como tercer paso se creó un índice de los embeddings usando FAISS (Facebook AI Similarity Search), una librería open source de META. Se decidió utilizar FAISS debido a que es una opción open source que puede funcionar localmente, evitando un potencial problema de privacidad, mientras sigue siendo eficiente y confiable (Douze et. al., 2025).

Código

A continuación, se muestra el código que fue utilizado para realizar los chunks y los embeddings. El archivo .ipynb completo se encuentra adjunto en la tarea como “embeddings.ipynb”.

```
!pip install openai faiss-cpu tiktoken
```

```
Requirement already satisfied: openai in /usr/local/lib/python3.12/dist-packages
(1.109.1)
Collecting faiss-cpu
  Downloading faiss_cpu-1.12.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.1 kB)
Requirement already satisfied: tiktoken in /usr/local/lib/python3.12/dist-
packages (0.11.0)
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.12/dist-
packages (from openai) (4.11.0)
Requirement already satisfied: distro<2,>=1.7.0 in
/usr/local/lib/python3.12/dist-packages (from openai) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.12/dist-packages (from openai) (0.28.1)
Requirement already satisfied: jiter<1,>=0.4.0 in /usr/local/lib/python3.12/dist-
packages (from openai) (0.11.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in
/usr/local/lib/python3.12/dist-packages (from openai) (2.11.9)
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages
(from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.12/dist-packages
(from openai) (4.67.1)
Requirement already satisfied: typing-extensions<5,>=4.11 in
/usr/local/lib/python3.12/dist-packages (from openai) (4.15.0)
Requirement already satisfied: numpy<3.0,>=1.25.0 in
/usr/local/lib/python3.12/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-
packages (from faiss-cpu) (25.0)
Requirement already satisfied: regex>=2022.1.18 in
/usr/local/lib/python3.12/dist-packages (from tiktoken) (2024.11.6)
Requirement already satisfied: requests>=2.26.0 in
/usr/local/lib/python3.12/dist-packages (from tiktoken) (2.32.4)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-
packages (from anyio<5,>=3.5.0->openai) (3.10)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages
(from httpx<1,>=0.23.0->openai) (2025.8.3)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-
packages (from httpx<1,>=0.23.0->openai) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-
packages (from httpcore==1.*->httpx<1,>=0.23.0->openai) (0.16.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=1.9.0->openai) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=1.9.0->openai)
(2.33.2)
```

```
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=1.9.0->openai) (0.4.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->tiktoken) (3.4.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->tiktoken) (2.5.0)
Downloading faiss_cpu-1.12.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (31.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 31.4/31.4 MB 81.5 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.12.0
```

```python
from openai import OpenAI
import faiss
import numpy as np
import tiktoken
import re
import os
import json
from google.colab import drive
from google.colab import userdata
```

```python
drive.mount('/content/drive')
API_KEY = userdata.get('MY_OPENAI_KEY')
```

```python
client = OpenAI(api_key= API_KEY)
```

```python
save_path = "/content/drive/MyDrive/RAG Recordings/"
os.makedirs(save_path, exist_ok=True)
```

```python
def chunk_text(text, max_tokens=400, model_name="text-embedding-3-small"):

    enc = tiktoken.encoding_for_model(model_name)

    sessions = re.split(r'---\s*.*?\s*---', text)

    chunks_with_meta = []

    for i in range(1, len(sessions), 2):
        session_id = sessions[i].strip()
        session_text = sessions[i+1].strip()

        sentences = re.split(r'(?<=[.!?]) +', session_text)
```

```
        current_chunk, current_len = [], 0

        for sentence in sentences:
            tokens = len(enc.encode(sentence))
            if current_len + tokens > max_tokens:
                chunks_with_meta.append({
                    "session_id": session_id,
                    "text": " ".join(current_chunk)
                })
                current_chunk, current_len = [sentence], tokens
            else:
                current_chunk.append(sentence)
                current_len += tokens

        if current_chunk:
            chunks_with_meta.append({
                "session_id": session_id,
                "text": " ".join(current_chunk)
            })

    return chunks_with_meta
```

```
text_path = "/content/drive/MyDrive/RAG Recordings/all_transcriptions.txt"
with open(text_path, "r", encoding="utf-8") as f:
    text = f.read()

chunks = chunk_text(text)
```

```
def get_embeddings(chunks, model="text-embedding-3-small"):
    embeddings = []
    for chunk in chunks:
        response = client.embeddings.create(
            model=model,
            input=chunk["text"]
        )
        embeddings.append(response.data[0].embedding)
    return np.array(embeddings, dtype="float32")

embeddings = get_embeddings(chunks)
```

```
dimension = embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
```

```
index.add(embeddings)

faiss_path = os.path.join(save_path, "recordings_index.faiss")
chunks_path = os.path.join(save_path, "recordings_chunks.json")

faiss.write_index(index, faiss_path)

with open(chunks_path, "w", encoding="utf-8") as f:
    json.dump(chunks, f, ensure_ascii=False, indent=2)
```

Conclusión

En esta fase se realizó el proceso para transformar las transcripciones generadas en la etapa previa en para que estuvieran listas en un vector database, siguiendo un proceso estructurado de acuerdo con lo que busca la metodología CRISP-ML. Se realizaron tres pasos clave: segmentación del texto en chunks coherentes, generación de embeddings con el modelo text-embedding-3-small, y la creación de un índice usando FAISS.

Lo que la metodología CRISP-ML busca en esta fase es que los datos queden limpios y estructurados a través de un proceso que sea reproducible y escalable. Eso es exactamente lo que se busco a la hora de tomar las decisiones de como es que se realizaría la separación de los chunks, sus embeddings, y la creación del índice de FAISS. Es decir, si llegaran nuevas transcripciones de más sesiones podrían entrar sin ningún problema en el pipeline para integrarse al resto de los datos de forma inmediata y en las mismas condiciones en que el resto de los datos fueron introducidos.

Además, un aspecto importante que se tomo en cuenta fue que cada proceso pudiera correr de manera independiente. Es decir, los chunks y los embeddings fueron creados en pasos separados, por lo que en caso de necesitar regresar a esta fase a ajustar el chunking o el modelo de embedding, esto pueda hacerse de manera sencilla.

Referencias:

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P., Lomeli M., Hosseini, L., J´egou, H. (2025). The FAISS Library. Meta. https://arxiv.org/pdf/2401.08281

OpenAI (2025). text-embedding-3-small. Model - OpenAI API