Instituto Tecnológico y de Estudios Superiores de Monterrey

TC5035.10 – Proyecto Integrador

Equipo 14

👥 Integrantes del equipo

- **Carlos Ricardo Álvarez Pérez**
  Matrícula: A01796116
- **Rodrigo Edgardo Armenta Santiago**
  Matrícula: A01795983
- **Susana Pérez Carranza**
  Matrícula: A01796151

MueblesRD - Document Ingestion Notebook

Este notebook permite ejecutar la ingesta de documentos PDF al vector store de Pinecone para el chatbot de MueblesRD.

Requisitos Previos

1. Cuenta de OpenAI con API key
2. Cuenta de Pinecone con API key e índice creado (`mueblesrd-index`)
3. (Opcional) Cuenta de LangSmith para tracing

Configuración de Secrets en Colab

1. Ir al icono 🔑 (Secrets) en el panel izquierdo
2. Añadir los siguientes secrets:
   - `OPENAI_API_KEY` (requerido)
   - `PINECONE_API_KEY` (requerido)
   - `LANGSMITH_API_KEY` (opcional)
   - `LANGSMITH_PROJECT` (opcional)
3. Habilitar "Notebook access" para cada uno

Celda 1: Instalación de Dependencias

```
!pip install langchain langchain-openai langchain-pinecone langchain-community pypdf pinecone-client
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.8/84.8 kB 3.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 37.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 329.1/329.1 kB 23.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 52.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 99.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 587.6/587.6 kB 38.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.7/64.7 kB 5.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.0/51.0 kB 3.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.3/259.3 kB 17.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.5/65.5 kB 4.9 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are insta
google-colab 1.0.0 requires requests==2.32.4, but you have requests 2.32.5 which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.4.1 which is incompatible.
prophet 1.2.2 requires numpy<2.4.0,>=1.15.4, but you have numpy 2.4.1 which is incompatible.
tensorflow 2.19.0 requires numpy<2.2.0,>=1.26.0, but you have numpy 2.4.1 which is incompatible.
```

Celda 2: Configuración de Variables de Entorno

```
from google.colab import userdata
import os

# Variables requeridas
os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
os.environ["PINECONE_API_KEY"] = userdata.get('PINECONE_API_KEY')

print("✅ Variables de entorno requeridas configuradas")

# Variables opcionales para LangSmith (tracing/monitoring)
try:
    os.environ["LANGSMITH_API_KEY"] = userdata.get('LANGSMITH_API_KEY')
    os.environ["LANGSMITH_PROJECT"] = userdata.get('LANGSMITH_PROJECT')
    os.environ["LANGSMITH_TRACING"] = "true"
    print("✅ LangSmith tracing habilitado")
except:
    print("ℹ️ LangSmith no configurado (opcional)")
```

```
✅ Variables de entorno requeridas configuradas
ℹ️ LangSmith no configurado (opcional)
```

Celda 3: Importaciones

```
import asyncio
import re
from typing import List

from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_core.documents import Document
from langchain_openai import OpenAIEmbeddings
from langchain_pinecone import PineconeVectorStore

print("✅ Importaciones completadas")
```

```
✅ Importaciones completadas
```

Celda 4: Funciones Auxiliares

```
# Section patterns to identify policy sections in the document
SECTION_PATTERNS = [
    r"(\d+\.?\d*\.-[A-Za-z\s]+)",  # Matches "0.-Global Procedure", "5.1 Validation"
    r"(\d+\.\s*[A-Z][A-Za-z\s]+(?:of|and|the|in|to|for|with)?[A-Za-z\s]*)",  # Matches "1. Verify Law
]

def extract_section_title(text: str) -> str:
    """Extract the section title from chunk content."""
    for pattern in SECTION_PATTERNS:
        match = re.search(pattern, text)
        if match:
            title = match.group(1).strip()
            # Clean up the title
            title = re.sub(r'\s+', ' ', title)
            if len(title) > 10:  # Ensure it's a meaningful title
                return title[:80]  # Limit length

    # Fallback: use first line if it looks like a header
    first_line = text.split('\n')[0].strip()
    if first_line and len(first_line) < 100 and not first_line.endswith('.'):
        return first_line[:80]

    return "MueblesRD Policy"

print("✅ Funciones auxiliares definidas")
```

```
✅ Funciones auxiliares definidas
```

Celda 5: Configuración de Embeddings y Pinecone

```
# Configuración del índice de Pinecone / Selecciona el index que hayas creado en Pinecone
INDEX_NAME = "mueblesrd-index-coolab"
```

```
# Parámetros de chunking optimizados para documentación de procedimientos
CHUNK_SIZE = 800
CHUNK_OVERLAP = 100
BATCH_SIZE = 50

# Inicializar embeddings
embeddings = OpenAIEmbeddings(
    model="text-embedding-3-small",
    show_progress_bar=True,
    chunk_size=50,
    retry_min_seconds=10
)

# Inicializar Pinecone vector store
vectorstore = PineconeVectorStore(
    index_name=INDEX_NAME,
    embedding=embeddings
)

print(f"✅ Embeddings y Pinecone configurados (índice: {INDEX_NAME})")
```

```
✅ Embeddings y Pinecone configurados (índice: mueblesrd-index-coolab)
```

Celda 6: Subir PDF

Ejecuta esta celda para subir tu archivo PDF desde tu computadora.

```
from google.colab import files

print("📁 Selecciona el archivo PDF a ingestar:")
uploaded = files.upload()

# Obtener el nombre del archivo subido
PDF_PATH = list(uploaded.keys())[0]
```

```
print(f"\n✅ Archivo subido: {PDF_PATH}")
print(f"   Tamaño: {len(uploaded[PDF_PATH]):,} bytes")
```

```
📁 Selecciona el archivo PDF a ingestar:
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving RD_POLITICS.pdf to RD_POLITICS (1).pdf

✅ Archivo subido: RD_POLITICS (1).pdf
   Tamaño: 283,495 bytes
```

Celda 7: Cargar y Procesar PDF

```
print("="*60)
print("📄 CARGA DE DOCUMENTO")
print("="*60)

# Cargar PDF
print(f"\n🔄 Cargando PDF: {PDF_PATH}")
loader = PyPDFLoader(PDF_PATH)
pages = loader.load()
print(f"✅ Cargadas {len(pages)} páginas")

# Mostrar información de cada página
print("\n📊 Información por página:")
for i, page in enumerate(pages):
    print(f"   Página {i + 1}: {len(page.page_content):,} caracteres")

print("\n" + "="*60)
print("✂️ FASE DE CHUNKING")
print("="*60)

# Dividir en chunks
print(f"\n🔄 Dividiendo en chunks (tamaño={CHUNK_SIZE}, overlap={CHUNK_OVERLAP})...")

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=CHUNK_SIZE,
```

```
    chunk_overlap=CHUNK_OVERLAP,
    separators=["\n\n", "\n", ". ", " ", ""],
)

chunks = text_splitter.split_documents(pages)

# Añadir metadatos a cada chunk
for chunk in chunks:
    section_title = extract_section_title(chunk.page_content)
    chunk.metadata["source"] = section_title
    chunk.metadata["file"] = PDF_PATH

# Estadísticas
unique_sections = set(c.metadata["source"] for c in chunks)
avg_chunk_size = sum(len(c.page_content) for c in chunks) // len(chunks)

print(f"\n✅ Creados {len(chunks)} chunks de {len(pages)} páginas")
print(f"   Secciones únicas identificadas: {len(unique_sections)}")
print(f"   Tamaño promedio de chunk: {avg_chunk_size} caracteres")

# Mostrar preview de chunks
print("\n📋 Preview de los primeros 3 chunks:")
for i, chunk in enumerate(chunks[:3]):
    print(f"\n--- Chunk {i+1} ---")
    print(f"Sección: {chunk.metadata['source']}")
    print(f"Página: {chunk.metadata.get('page', 'N/A')}")
    print(f"Contenido: {chunk.page_content[:150]}...")
```

```
============================================================
📄 CARGA DE DOCUMENTO
============================================================

🔄 Cargando PDF: RD_POLITICS (1).pdf
✅ Cargadas 9 páginas

📊 Información por página:
   Página 1: 2,851 caracteres
   Página 2: 2,847 caracteres
   Página 3: 2,445 caracteres
   Página 4: 2,710 caracteres
```

```
   Página 5: 2,907 caracteres
   Página 6: 3,352 caracteres
   Página 7: 2,735 caracteres
   Página 8: 2,679 caracteres
   Página 9: 2,088 caracteres

============================================================
✂️ FASE DE CHUNKING
============================================================

🔄 Dividiendo en chunks (tamaño=800, overlap=100)...

✅ Creados 37 chunks de 9 páginas
   Secciones únicas identificadas: 32
   Tamaño promedio de chunk: 721 caracteres

📋 Preview de los primeros 3 chunks:

--- Chunk 1 ---
Sección: 0.-Global Procedure
Página: 0
Contenido: 0.-Global Procedure - Admissibility of a Request

Purpose: This is a global procedure intended to validate the admissibility of a request. The
proc...

--- Chunk 2 ---
Sección: 3. Verification of Request Admissibility This section includes several sub
Página: 0
Contenido: To ensure that only one request is processed for a specific issue, it is necessary to checl
duplicates in Salesforce. Consult the training: Dupli...

--- Chunk 3 ---
Sección: 5. Respecting Deadlines To ensure the customer
Página: 0
Contenido: Contract Number to follow the steps.
5. Respecting Deadlines
To ensure the customer's request complies with the granted timeframes, it is important ...
```

Celda 8: Indexar en Pinecone

Esta celda indexa todos los chunks en Pinecone.

```
async def index_documents_async(documents: List[Document], batch_size: int = BATCH_SIZE):
    """Process documents in batches asynchronously"""
    print("="*60)
    print("📤 FASE DE INDEXACIÓN")
    print("="*60)
    print(f"\n🔄 Indexando {len(documents)} documentos en Pinecone ({INDEX_NAME})...")
    print(f"   Tamaño de batch: {batch_size}")

    # Crear batches
    batches = [
        documents[i:i + batch_size] for i in range(0, len(documents), batch_size)
    ]

    print(f"   Total de batches: {len(batches)}")
    print()

    async def aadd_batch(batch: List[Document], batch_number: int):
        try:
            await vectorstore.aadd_documents(batch)
            print(f"   ✅ Batch {batch_number}/{len(batches)}: {len(batch)} documentos indexados")
            return True
        except Exception as e:
            print(f"   ❌ Batch {batch_number}: Error - {str(e)}")
            return False

    # Procesar batches concurrentemente
    tasks = [aadd_batch(batch, i + 1) for i, batch in enumerate(batches)]
    results = await asyncio.gather(*tasks, return_exceptions=True)

    # Contar batches exitosos
    success_count = sum(1 for result in results if result is True)
```

```
        print()
        if success_count == len(batches):
            print("✅ Todos los batches indexados exitosamente.")
        else:
            print(f"⚠️ {success_count}/{len(batches)} batches indexados exitosamente.")

        return success_count == len(batches)

    # Ejecutar la indexación
    await index_documents_async(chunks, batch_size=BATCH_SIZE)

    # Resumen final
    print("\n" + "="*60)
    print("📊 RESUMEN DE INGESTA")
    print("="*60)
    print(f"   Archivo: {PDF_PATH}")
    print(f"   Páginas procesadas: {len(pages)}")
    print(f"   Chunks creados: {len(chunks)}")
    print(f"   Tamaño de chunk: {CHUNK_SIZE} caracteres")
    print(f"   Overlap: {CHUNK_OVERLAP} caracteres")
    print(f"   Índice destino: {INDEX_NAME}")
    print("\n✅ Proceso de ingesta completado.")
```

```
============================================================
📤 FASE DE INDEXACIÓN
============================================================

🔄 Indexando 37 documentos en Pinecone (mueblesrd-index-coolab)...
   Tamaño de batch: 50
   Total de batches: 1

  0%                                                 0/1 [00:00<?, ?it/s]
   ✅ Batch 1/1: 37 documentos indexados

✅ Todos los batches indexados exitosamente.

============================================================
📊 RESUMEN DE INGESTA
============================================================
   Archivo: RD_POLITICS (1).pdf
   Páginas procesadas: 9
   Chunks creados: 37
   Tamaño de chunk: 800 caracteres
   Overlap: 100 caracteres
   Índice destino: mueblesrd-index-coolab

✅ Proceso de ingesta completado.
```

Celda 9: Prueba de Búsqueda de Similitud

Usa esta celda para probar búsquedas en el vector store y verificar que la ingesta funcionó correctamente.

```
# Texto de consulta para probar
query = input("🔍 Ingresa tu consulta de prueba: ")

print(f"\n" + "="*60)
print("🔎 RESULTADOS DE BÚSQUEDA")
print("="*60)
print(f"\nConsulta: \"{query}\"\n")
```

```
# Realizar búsqueda de similitud con scores
results = vectorstore.similarity_search_with_score(query, k=3)

if not results:
    print("❌ No se encontraron resultados.")
else:
    print(f"✅ Se encontraron {len(results)} resultados:\n")

    for i, (doc, score) in enumerate(results, 1):
        print(f"--- Resultado {i} (Score: {score:.4f}) ---")
        print(f"📌 Sección: {doc.metadata.get('source', 'N/A')}")
        print(f"📄 Página: {doc.metadata.get('page', 'N/A')}")
        print(f"🗂️ Archivo: {doc.metadata.get('file', 'N/A')}")
        print(f"\n📝 Contenido:")
        print(f"{doc.page_content[:500]}..." if len(doc.page_content) > 500 else doc.page_content)
        print("\n" + "-"*60 + "\n")
```

```
🔍 Ingresa tu consulta de prueba: LG aesthetic

============================================================
✅ RESULTADOS DE BÚSQUEDA
============================================================
Consulta: "LG aesthetic"
```

Celda 10: Búsquedas Adicionales (Opcional)

Ejecuta múltiples búsquedas sin necesidad de volver a ingresar cada vez.

```
def search_vectorstore(query: str, k: int = 3):
    """Función auxiliar para búsquedas rápidas"""
    print(f"\n🔍 Buscando: \"{query}\"\n")
    results = vectorstore.similarity_search_with_score(query, k=k)

    for i, (doc, score) in enumerate(results, 1):
        print(f"[{i}] Score: {score:.4f} | Sección: {doc.metadata.get('source', 'N/A')[:40]}")
        print(f"    {doc.page_content[:200]}...\n")

    return results

# Ejemplos de uso:
# search_vectorstore("duration")
```