Maestría en Inteligencia Artificial Aplicada

Curso: Proyecto Integrador

Avance 1. Análisis exploratorio de datos

A01796116 Carlos Ricardo Álvarez Pérez
A01795983 Rodrigo Edgardo Armenta Santiago
A01796151 Susana Pérez Carranza

1 de febrero de 2026

Introducción

En esta fase de un proyecto de aprendizaje automática, típicamente se realiza el análisis exploratorio de datos. Sin embargo, por la naturaleza de este proyecto (como se describe en el segundo entregable), en este caso, se realizó el estudio de datos no estructurados y su procesamiento para ser convertido en *embeddings*.

Además, en relación con la dimensión de desarrollo de software, se realizó la planeación del producto bajo un marco de trabajo *Scrum* y la estructuración del código fuente en un repositorio que permita el control de versiones.

En esta entrega del proyecto integrador se muestra la planeación scrum y el trabajo desarrollado en el primer sprint.

Planeación del product backlog

El Product Backlog es una lista priorizada y dinámica de todo el trabajo necesario para desarrollar un producto que entregue valor, incluyendo funcionalidades, mejoras, correcciones y requisitos técnicos. En el contexto de un Producto Mínimo Viable (MVP), el backlog representa el conjunto de elementos esenciales para validar la propuesta de valor del producto.

Los elementos del backlog suelen organizarse inicialmente en épicas (epics), que se descomponen en historias de usuario. Estas historias describen funcionalidades desde la perspectiva del usuario final, con el objetivo de que el equipo de desarrollo comprenda el propósito de cada funcionalidad y pueda empatizar con las necesidades reales del usuario.

Durante la planificación del sprint, el equipo de desarrollo selecciona de forma colaborativa las historias de usuario que considera factibles de completar en el sprint, basándose en su capacidad y experiencia. A partir de estas historias, el equipo define de manera autónoma las tareas técnicas necesarias para su implementación. Una vez seleccionadas, el equipo se compromete a alcanzar los objetivos definidos para ese sprint.

A continuación se muetran las épicas e historias de usuario generadas durante la primera sesión de planeación de producto.

Base de Conocimiento

Este épico involucra la creación de una base de conocimiento que el bot pueda utilizar para enviar recomendaciones.

Preprocesamiento de Tickets de Ejemplo

"Como nuevo agente de atención al cliente, quiero que el copiloto QA aproveche los tickets reales proporcionados por Meubles RD, para que las recomendaciones y validaciones que reciba estén alineadas con los casos reales que manejo y me ayuden a evitar errores comunes desde el principio."

Criterios de Aceptación:

- Contamos con una tabla con los campos esperados del ticket, y cada fila representa un ticket de ejemplo. Esperamos un tamaño de muestra de entre 60 y 100 tickets.

Definición de Hecho (DoD):

- El sistema backend es capaz de recibir los datos de los tickets de acuerdo con los ejemplos analizados.

Demostración de Fragmentos (Snippets)

"Como agente, quiero que el copiloto me muestre fragmentos relevantes de los Procedimientos Operativos Estándar (SOP) basados en el tipo de reclamo, para asegurar que sigo el proceso correcto."

Criterios de Aceptación:

- El copiloto debe mostrar automáticamente fragmentos de SOP relacionados con el tipo de reclamo (daño o pieza faltante).
- El contenido sugerido debe estar alineado con el contenido visible en el ticket.
- El agente puede ver el fragmento completo o expandirlo si es necesario.

DoD:

- Los SOP están indexados en la base de datos vectorial.
- El copiloto puede consultar la base de datos y recuperar fragmentos relevantes.
- Probado con al menos 3 tipos de reclamos distintos.
- Validado por un usuario de negocio o experto en SOP.
- La prueba se realiza en la demostración de la extensión.

Guías de procedimientos operativos estándar (SOP)

Este épico incluye todo lo relacionado para que el copiloto tenga todas las funciones para proporcionar orientación de acuerdo con las guías de SOP.

Búsqueda de Recordatorios y Validaciones

Criterios de Aceptación:

- El copiloto debe validar que los campos requeridos según las políticas de RD Furniture estén completos.
- El copiloto debe alertarme si faltan pruebas (como fotos) según el tipo de reclamo.
- El copiloto debe mostrar mensajes explicativos cuando se aplica una política específica (por ejemplo, verificación de identidad según la Ley 25).

DoD:

- El endpoint backend debe estar disponible.
- Al enviar los datos del ticket, debe responder con los resultados de la validación, indicando si los campos requeridos están completos según el tipo de reclamo, si existen las pruebas necesarias y devolver mensajes explicativos según las políticas.

Campos Faltantes Detectados

"Como agente, quiero que el copiloto indique los campos requeridos faltantes según el tipo de reclamo, para que pueda completarlos antes de continuar."

Criterios de Aceptación:

- El copiloto debe identificar los campos requeridos faltantes (por ejemplo, número de pedido, ID de producto, número de serie).
- La validación debe considerar el tipo de reclamo (daño vs. pieza faltante).
- El mensaje debe ser claro y contextual (por ejemplo, "El número de serie es obligatorio para este tipo de producto").

DoD:

- El backend expone un endpoint que recibe los datos del ticket y responde indicando si los campos requeridos están completos según el tipo de reclamo.

Sugerencias de Tono

“Como agente de atención al cliente, quiero que el sistema indique si un reclamo debe enrutarse a un proveedor específico según las reglas de negocio, para que pueda tomar decisiones informadas y brindar una mejor experiencia al cliente.”

Criterios de Aceptación:

- El copiloto debe identificar cuando un cliente se encuentra en un estado emocional sensible, como enojo o frustración, según los datos del ticket.
- La respuesta proporcionada debe sugerir el uso de un tono empático y comprensivo al leer los tickets de clientes frustrados o enojados.

DoD:

- Cuando la API recibe datos de tickets de clientes en este tipo de situación, sugiere el estilo de respuesta descrito.

Prueba Fotográfica

“Como agente de atención al cliente, quiero que se me alerte cuando falte la prueba fotográfica requerida, para que pueda evitar rechazos y retrabajos.”

Criterios de Aceptación:

- El copiloto detecta la falta de fotos para los tipos de reclamos que las requieren (por ejemplo, producto dañado).
- El mensaje explica claramente el requisito de la foto ("Se requiere al menos una foto para validar el daño reportado").

DoD:

- El endpoint backend evalúa la prueba fotográfica según el tipo de reclamo.

3 Principios de la empresa (GAC)

"Como agente de atención al cliente, necesito que se me asesore de acuerdo con los 3 principios clave de GAC, para que cumpla adecuadamente con los estándares de comunicación de MRD."

Criterios de Aceptación:

- Basado en el tipo de reclamo y el contenido escrito por el cliente.
- Longitud, estructura y tono del mensaje (por ejemplo, todo en mayúsculas, puntuación repetida, lenguaje emocional).
- La respuesta proporcionada cumple con los 3 principios de GAC (propiedad radical, solución a través de opciones, creación de valor a través de la anticipación de necesidades/problemas futuros).

DoD:

- Las respuestas proporcionadas cumplen con estos criterios. GAC prueba el modelo y valida las respuestas.

Extensión de Chrome

Este épico incluye todo lo relacionado para tener lista la extensión de Chrome.

Maqueta de la Extensión de Chrome

"Como agente de servicio, necesito que la interfaz del chatbot esté integrada en mi navegador para poder usarla dentro de la vista de reclamos de Salesforce."

Criterios de Aceptación:

- El chatbot debe estar disponible como una extensión de Chrome.
- Esta historia cubre la creación de una extensión de Chrome con un diseño básico del chatbot, sin necesidad de conexión a un servicio en esta etapa.

DoD:

- Tenemos los archivos fuente de la extensión con un archivo MD simple que explica cómo instalarla en Chrome o Edge. Al instalarla, la interfaz se muestra con la paleta de colores de Meubles RD, pero sin ninguna funcionalidad aún.

Activación del Copiloto

"Como agente, necesito un botón en el navegador para poder hacer clic y recibir sugerencias del bot."

Esta historia implica agregar un nuevo botón al navegador al instalar la extensión de Chrome, con el icono del bot (potencialmente el icono proporcionado por MeublesRD). Al hacer clic en el botón, se capturarán los datos del ticket abierto, se enviarán al backend, se esperará una respuesta y se mostrarán las recomendaciones.

Criterios de Aceptación:

- Al hacer clic, la extensión debe poder detectar si la vista actual no es un ticket.
- Mientras se procesan los datos, debe mostrarse un indicador de carga (por ejemplo, un spinner de carga).
- Después de esperar, la extensión muestra la respuesta proporcionada por el backend.

DoD:

- Una vez instalada la extensión y activa pero aún "cerrada", al hacer clic, el frontend envía la solicitud al backend mostrando un indicador de carga.

Extracción de Campos del Ticket

"Como agente de servicio, quiero que el copiloto analice automáticamente la información visible en mi pantalla (campos del ticket, comentarios, archivos adjuntos), para que pueda recibir orientación contextual sin necesidad de ingresar datos manualmente."

Criterios de Aceptación:

- Esta historia implica la implementación de un mecanismo de lectura del DOM o un scraping controlado desde la extensión del navegador.
- Debe adherirse a los límites de privacidad y seguridad definidos para la demostración.
- Esto se puede dividir en subtareas técnicas: extracción de campos, extracción de texto, detección de archivos adjuntos, etc.

DoD:

- Al hacer clic en el botón de la extensión, los campos definidos por el equipo para ser procesados se toman y se envían al backend.

Comprensión del Ticket

Este épico incluye todas las actividades para permitir que el copiloto comprenda los tickets y las necesidades del cliente.

Confirmación del Tipo de Ticket

"Como agente de servicio, necesito saber si el chatbot puede ayudar con el ticket actual, para asegurarme de poder confiar en sus sugerencias."

Criterios de Aceptación:

- El chatbot debe reconocer si el ticket se relaciona con reclamos relacionados con artículos dañados y piezas faltantes.
- El modelo debe ingerir los datos disponibles y determinar el tipo de reclamo.

DoD:

- El sistema backend es capaz de recibir los datos del ticket y determinar el tipo de reclamo.

Backend REST: Especificaciones y Criterios de Aceptación

Este épico engloba todas las actividades necesarias para desarrollar el backend REST, el cual recibirá llamadas desde el frontend y orquestará las consultas a la base de conocimiento y a las APIs de LLM (Large Language Model).

Mockup de la API REST

"Como agente de atención al cliente, quiero recibir recomendaciones contextuales en tiempo real mientras gestiono una reclamación, para reducir errores y asegurar que sigo los procedimientos correctamente."

Criterios de Aceptación:

- El backend básico está construido y disponible en un repositorio.
- Está en funcionamiento y listo para proporcionar respuestas en tiempo real.
- Implementa protecciones de seguridad mínimas (autenticación básica de API).
- Expone endpoints para realizar consultas.

DoD:

- Existe un REST en funcionamiento en un backend accesible para los miembros del equipo del proyecto.

Recuperación de Respuestas del Backend

"Como agente de atención al cliente, quiero que el copiloto muestre mensajes explicativos cuando se aplique una política específica, para comprender el razonamiento detrás de cada requisito."

Criterios de Aceptación:

- El copiloto muestra mensajes cuando se detecta una política aplicable (ej., "Se requiere verificación de identidad según la Ley 25").
- Los mensajes deben ser breves, claros y contextualizados a la reclamación.
- Una única fuente de verdad (base de conocimiento) respalda estos mensajes.

DoD:

- El backend expone un endpoint que, al recibir datos del ticket, devuelve mensajes explicativos relevantes basados en las políticas de Meubles RD.

Validación del Formato de Datos

"Como agente de atención al cliente, quiero que el sistema indique si algún dato tiene un formato incorrecto (ej., un número de serie inválido), para poder corregirlo inmediatamente y cumplir con los requisitos del proceso."

Criterios de Aceptación:

- El backend confirma el formato de los datos de acuerdo con la base de conocimiento.
- El endpoint puede recibir datos del ticket y devolver advertencias cuando un campo de datos tiene un formato incorrecto.

DoD:

- Se realiza una prueba en modo API o script, el backend recibe datos del ticket y advierte sobre formatos incorrectos o confirma que los campos están bien formateados.

Entrega de Recordatorios y Validaciones

"Como agente, quiero que el copiloto me proporcione recordatorios y validaciones basados en los PSOs (Procedimientos Operativos Estándar), para asegurar que sigo los procedimientos correctamente y evitar errores que puedan afectar al cliente o al cumplimiento normativo."

Criterios de Aceptación:

- La base de conocimiento contiene todos los pasos y validaciones.
- El backend realiza una verificación cruzada del proceso a seguir versus los PSOs.

DoD:

- La respuesta proporcionada recuerda al agente los pasos a seguir y las validaciones a realizar.

Proceso de Generación de Embeddings para el Sistema RAG de Muebles RD

¿Qué es RAG?

RAG (Retrieval-Augmented Generation) es una arquitectura que combina dos componentes principales:

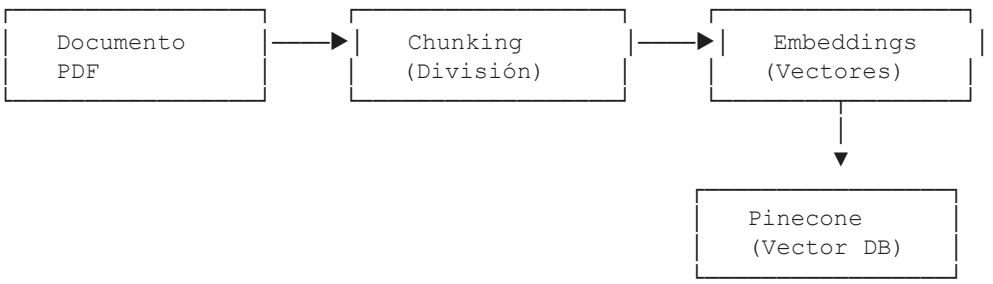
- Recuperación (Retrieval): Búsqueda de documentos relevantes en una base de conocimiento vectorial.
- Generación (Generation): Uso de un modelo de lenguaje (LLM) para generar respuestas basadas en los documentos recuperados.

¿Qué son los Embeddings?

Los embeddings son representaciones vectoriales de texto en un espacio de alta dimensionalidad. Cada texto se convierte en un vector numérico donde textos semánticamente similares tienen vectores cercanos entre sí. Esto permite realizar búsquedas por similitud semántica en lugar de búsquedas por palabras clave exactas.

Arquitectura del Sistema de Ingesta de Datos

El sistema MueblesRD Helper utiliza la siguiente arquitectura para procesar documentos:



Componentes Utilizados

Componente	Función
PyPDFLoader	Carga y extracción de texto de documentos PDF
RecursiveCharacterTextSplitter	División del texto en fragmentos manejables
OpenAIEmbeddings	Generación de vectores usando el modelo text-embedding-3-small
PineconeVectorStore	Almacenamiento y búsqueda de vectores

Proceso de Ingesta de Documentos

Configuración Inicial

El primer paso consiste en importar las bibliotecas necesarias y configurar las conexiones a los servicios externos (OpenAI y Pinecone).

```
import asyncio
import re
from typing import List

from dotenv import load_dotenv
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

# Cargar variables de entorno
load_dotenv()
```

Las credenciales de API se almacenan en un archivo .env para mantener la seguridad:

```
OPENAI_API_KEY=sk-...
PINECONE_API_KEY=...
LANGSMITH_API_KEY=...
```

Carga del Documento PDF

Se utiliza PyPDFLoader de LangChain para cargar el documento PDF. Este componente extrae el texto de cada página y lo convierte en objetos Document que contienen el contenido y metadatos asociados.

```
# Ruta del archivo PDF
PDF_PATH = "RD_POLITICS.pdf"

# Cargar el documento PDF
loader = PyPDFLoader(PDF_PATH)
pages = loader.load()

print(f"Se cargaron {len(pages)} páginas del PDF")

# Cada página es un objeto Document con:
# - page_content: El texto extraído
# - metadata: Información adicional (número de página, fuente, etc.)
```

El resultado es una lista de objetos Document, donde cada objeto representa una página del PDF original con su contenido textual y metadatos.

División del Texto (Chunking)

Los modelos de embeddings tienen límites en la cantidad de texto que pueden procesar. Además, fragmentos más pequeños permiten una recuperación más precisa. Por estas razones, el texto se divide en "chunks" o fragmentos más pequeños.

Parámetros de Configuración

chunk_size = 800: Tamaño máximo de cada fragmento en caracteres. Un valor de 800 es apropiado para documentos procedimentales donde cada sección es relativamente corta.

chunk_overlap = 100: Cantidad de caracteres que se superponen entre fragmentos consecutivos. Esto asegura que no se pierda contexto en los límites.

separators: Lista de caracteres usados para dividir el texto, en orden de prioridad: párrafos (\n\n), saltos de línea (\n), puntos (.), espacios, etc.

```
# Configuración del text splitter
CHUNK_SIZE = 800
CHUNK_OVERLAP = 100

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=CHUNK_SIZE,
    chunk_overlap=CHUNK_OVERLAP,
    separators=["\n\n", "\n", ". ", " ", ""],  # Prioriza límites de párrafo
)

# Dividir los documentos en chunks
chunks = text_splitter.split_documents(pages)

print(f"Se crearon {len(chunks)} fragmentos de {len(pages)} páginas")
```

3.3.2 Funcionamiento del RecursiveCharacterTextSplitter

Este splitter funciona de manera recursiva:

1. Intenta dividir el texto usando el primer separador (\n\n para párrafos).

2. Si los fragmentos resultantes son mayores que chunk_size, usa el siguiente separador.

3. Continúa recursivamente hasta que todos los fragmentos cumplan con el tamaño máximo.

4. Añade overlap entre fragmentos para mantener continuidad semántica.

Extracción de Metadatos de Sección

Para mejorar la trazabilidad de las respuestas, se extraen los títulos de sección del contenido de cada chunk usando expresiones regulares:

```
# Patrones para identificar secciones en el documento
SECTION_PATTERNS = [
    r"(\d+\.?\d*\.-[A-Za-z\s]+)",  # Ej: "0.-Global Procedure"
    r"(\d+\.\s*[A-Z][A-Za-z\s]+)",  # Ej: "1. Verify Law 25"
]

def extract_section_title(text: str) -> str:
    """Extrae el título de sección del contenido del chunk."""
    for pattern in SECTION_PATTERNS:
        match = re.search(pattern, text)
        if match:
            title = match.group(1).strip()
            title = re.sub(r'\s+', ' ', title)
            if len(title) > 10:
                return title[:80]

    # Fallback: usar la primera línea si parece un encabezado
    first_line = text.split('\n')[0].strip()
    if first_line and len(first_line) < 100:
        return first_line[:80]

    return "MueblesRD Policy"

# 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
```

Generación de Embeddings

Los embeddings se generan utilizando el modelo text-embedding-3-small de OpenAI. Este modelo convierte texto en vectores de 1536 dimensiones.

```
# Inicializar el modelo de embeddings
embeddings = OpenAIEmbeddings(
    model="text-embedding-3-small",
    show_progress_bar=True,
    chunk_size=50,        # Procesar 50 textos por llamada API
    retry_min_seconds=10  # Reintentar después de 10 segundos si falla
)

# Inicializar el vector store de Pinecone
vectorstore = PineconeVectorStore(
    index_name="mueblesrd-index",
    embedding=embeddings
)
```

Características del Modelo text-embedding-3-small

Característica	Valor
Dimensionalidad	1536 dimensiones
Contexto máximo	8191 tokens
Normalización	Los vectores están normalizados (longitud = 1)
Similitud	Se usa similitud coseno para comparar vectores

Almacenamiento en Pinecone

Los vectores generados se almacenan en Pinecone, una base de datos vectorial optimizada para búsquedas de similitud a gran escala. El proceso se realiza de forma asíncrona en lotes para mayor eficiencia.

```
BATCH_SIZE = 50  # Tamaño del lote para indexación

async def index_documents_async(documents: List[Document], batch_size: int):
    """Procesa documentos en lotes de forma asíncrona."""

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

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

    async def process_batch(batch: List[Document], batch_num: int):
        try:
            # Añadir documentos al vector store
            # Internamente, esto:
            # 1. Genera embeddings para cada documento
            # 2. Sube los vectores a Pinecone con sus metadatos
            await vectorstore.aadd_documents(batch)
            print(f"Lote {batch_num}: Indexados {len(batch)} documentos")
            return True
        except Exception as e:
            print(f"Lote {batch_num}: Error - {str(e)}")
            return False

    # Procesar todos los lotes concurrentemente
    tasks = [process_batch(batch, i+1) for i, batch in enumerate(batches)]
    results = await asyncio.gather(*tasks)

    success = sum(1 for r in results if r is True)
    print(f"{success}/{len(batches)} lotes indexados exitosamente")
```

Estructura de Datos en Pinecone

Cada registro en Pinecone contiene:

- id: Identificador único generado automáticamente
- values: Vector de 1536 dimensiones (el embedding)
- metadata: Diccionario con información adicional (source, file, page, etc.)

Proceso de Recuperación (Retrieval)

Una vez que los documentos están indexados, el sistema puede recuperar información relevante para responder preguntas de los usuarios.

```
@tool(response_format="content_and_artifact")
def retrieve_context(query: str):
    """Recupera documentación relevante para responder consultas."""

    # 1. La query del usuario se convierte en un embedding
    # 2. Se buscan los vectores más similares en Pinecone
    # 3. Se retornan los documentos correspondientes

    retrieved_docs = vectorstore.as_retriever().invoke(query, k=4)

    # Serializar documentos para el modelo
    serialized = "\n\n".join(
        f"Source: {doc.metadata.get('source', 'Unknown')}\n"
        f"Content: {doc.page_content}"
        for doc in retrieved_docs
    )

    return serialized, retrieved_docs
```

Similitud Coseno

La búsqueda se realiza usando similitud coseno, que mide el ángulo entre dos vectores. Valores cercanos a 1 indican alta similitud semántica:

```
similitud(A, B) = (A · B) / (||A|| × ||B||)

Donde:
- A · B es el producto punto de los vectores
- ||A|| y ||B|| son las magnitudes de los vectores
```

Flujo Completo de Ejecución

```
async def main():
    """Función principal de ingesta de documentos."""

    print("Iniciando proceso de ingesta...")

    # 1. Cargar PDF
    loader = PyPDFLoader(PDF_PATH)
    pages = loader.load()
    print(f"Cargadas {len(pages)} páginas")

    # 2. Dividir en chunks
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=CHUNK_SIZE,
        chunk_overlap=CHUNK_OVERLAP,
        separators=["\n\n", "\n", ". ", " ", ""]
    )
    chunks = text_splitter.split_documents(pages)
    print(f"Creados {len(chunks)} fragmentos")

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

    # 4. Indexar en Pinecone (genera embeddings automáticamente)
    await index_documents_async(chunks, batch_size=BATCH_SIZE)

    print("Proceso completado exitosamente")
    print(f"  - Páginas procesadas: {len(pages)}")
    print(f"  - Chunks creados: {len(chunks)}")
    print(f"  - Índice: mueblesrd-index")

# Ejecutar
if __name__ == "__main__":
    asyncio.run(main())
```

Consideraciones Técnicas

Selección del Tamaño de Chunk

La elección del tamaño de chunk (800 caracteres) se basa en:

- Naturaleza del documento: Procedimientos con pasos numerados y secciones definidas.
- Balance entre contexto y precisión: Chunks muy grandes reducen la precisión de búsqueda.
- Límites del modelo: El modelo de embeddings puede procesar hasta 8191 tokens.
- Costos: Chunks más pequeños generan más vectores pero permiten respuestas más específicas.

Overlap entre Chunks

El overlap de 100 caracteres asegura que:

- No se pierda información en los límites entre chunks.
- El contexto fluya naturalmente entre fragmentos consecutivos.
- Las búsquedas puedan encontrar información que cruza límites de chunks.

Procesamiento Asíncrono

El uso de asyncio permite procesar múltiples lotes en paralelo, reduciendo significativamente el tiempo total de indexación para documentos grandes.

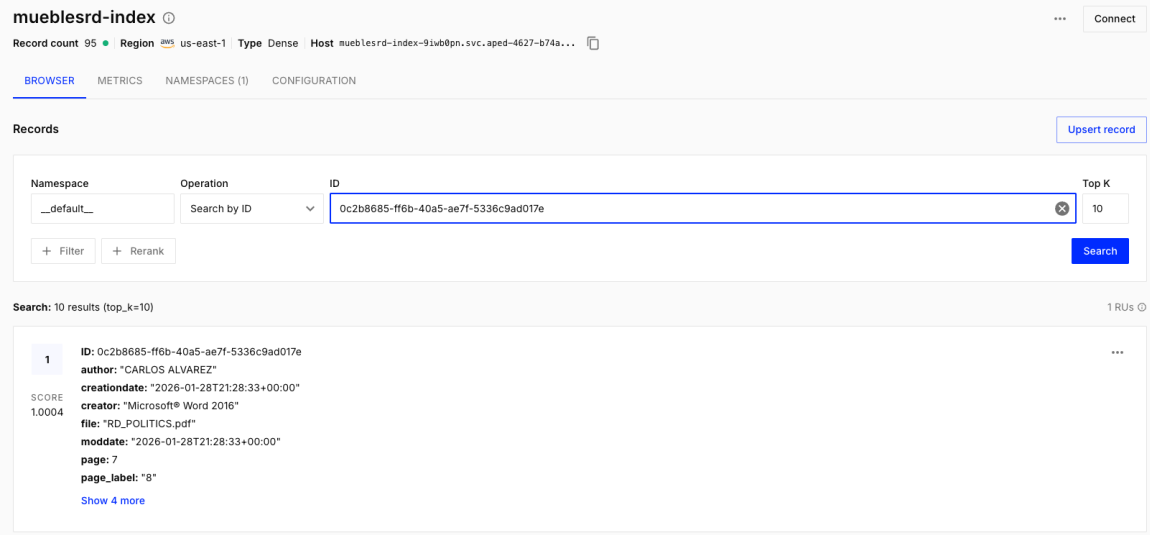


Fig 6. Almacenamiento de embeddings en Pinecone para su fácil recuperación.

Conclusiones

Se ha hecho la planeación general del producto bajo un marco de trabajo scrum, lo que permite entregar valor a la empresa al mismo tiempo enfocándose en el cliente. En las siguientes etapas trabajaremos sobre el *product backlog* para irlo refinando y priorizando.

El proceso de generación de embeddings transforma documentos de texto en representaciones vectoriales que permiten búsquedas semánticas eficientes. Los pasos clave son:

1. Carga del documento fuente (PDF)
2. División en fragmentos manejables (chunking)
3. Extracción de metadatos para trazabilidad
4. Generación de vectores usando modelos de embeddings
5. Almacenamiento en una base de datos vectorial
6. Recuperación por similitud semántica

Este pipeline permite que el sistema RAG responda preguntas de los usuarios basándose en el contenido real de los documentos de la empresa, proporcionando respuestas precisas y citando las fuentes relevantes.

Referencias bibliográficas

- Proyectos ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos. 2015. Martín Alaimo, Martín Salías.
- Sutherland, Jeff. (2021). Scrum: El Arte de Hacer el Doble de Trabajo en la Mitad de TiempoLinks to an external site., Océano de México