

RAG para interactuar con tu artículo científico, versión gratuita

CÉSAR AUGUSTO PILÓN ALCALÁ

1. Objetivo

2

- ▶ Leer un artículo científico.
- ▶ Responder preguntas usando solo la información del artículo.
- ▶ Mantener respuestas concisas o detalladas según lo necesitemos.

2. Herramientas

- ▶ LangChain → Para construir cadenas de recuperación y generación (RAG).
- ▶ Chroma → Base de datos vectorial local, gratuita.
- ▶ HuggingFace Embeddings → Convierte textos en vectores semánticos (gratis).
- ▶ Ollama (opcional) o cualquier LLM local → Para generar respuestas a partir de los fragmentos recuperados.

3. Flujo de trabajo

4

Usuario → Pregunta



Retriever (Chroma)



Fragmentos relevantes



LLM (Ollama/HF)



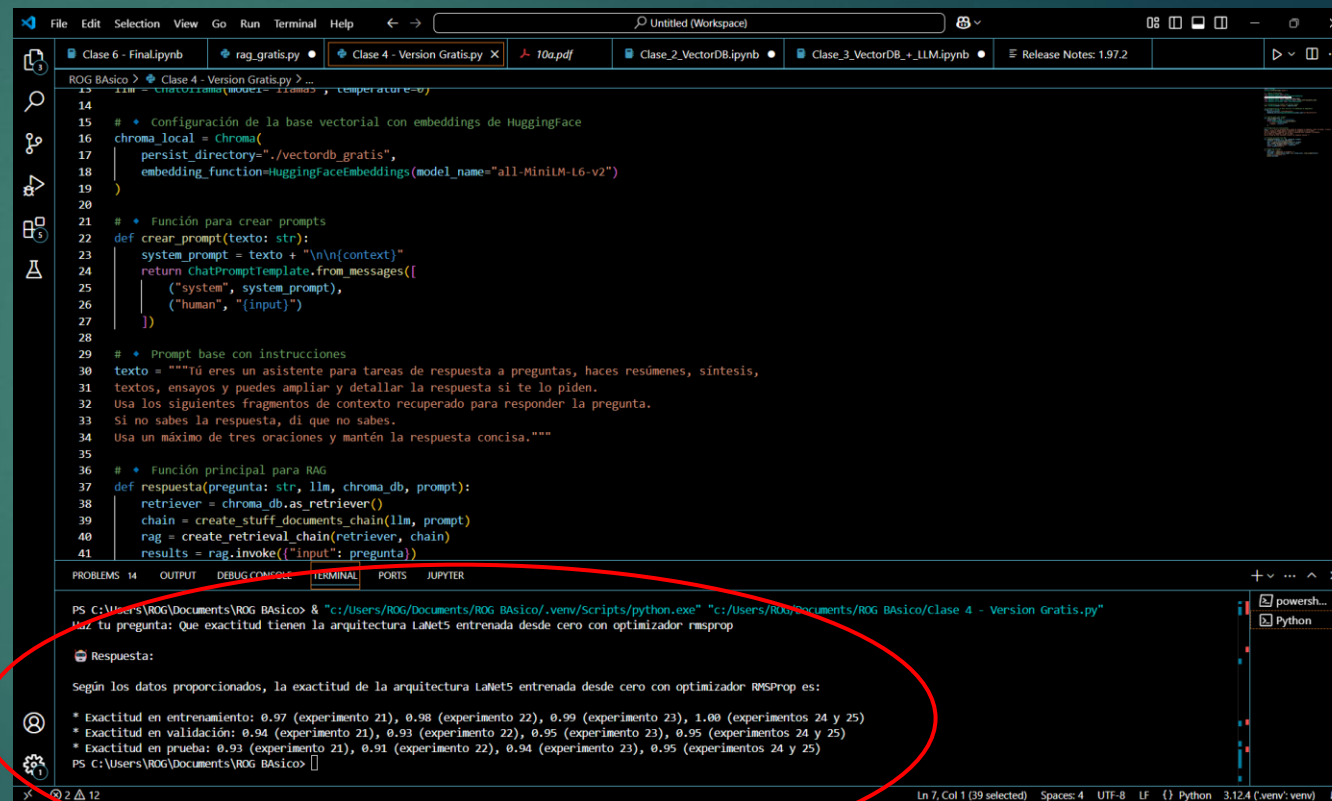
Respuesta

Resultados

5

CONCLUSIONES

Se determinó que la red neuronal LeNet5 entrenada desde cero, con optimizador RMSProp y tasa de aprendizaje 1×10^{-4} logra clasificar correctamente manzanas sanas y dañadas de las variedades Red Delicious, Granny Smith, Golden Delicious y Gala, con una exactitud del 97 %; sin embargo, considerando la matriz de confusión, se puede inferir que esta arquitectura funciona al 100 % para clasificar las categorías Gala sana, Golden dañada y Granny Smith sana; para las cinco categorías restantes, el porcentaje varía entre 92 y 97 %, lo cual es importante porque la norma oficial mexicana (SE, 2003) permite que hasta 10 % de las manzanas de cualquier lote no reúna los requisitos enunciados. Para fines de una implementación que considere cumplir con la norma oficial mexicana NMX-FF-061-SCFI-2003 la arquitectura encontrada funciona. Un hallazgo en este trabajo fue que el diseño experimental



```
File Edit Selection View Go Run Terminal Help
ROG BASICO > Class 4 - Version Gratis.py >
13 llm = ChatOpenAI(model="gpt-4o", temperature=0)
14
15 # Configuración de la base vectorial con embeddings de HuggingFace
16 chroma_local = Chroma(
17     persist_directory="./vectordb_gratis",
18     embedding_function=HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
19 )
20
21 # Función para crear prompts
22 def crear_prompt(texto: str):
23     system_prompt = texto + "\n\n(context)"
24     return ChatPromptTemplate.from_messages([
25         ("system", system_prompt),
26         ("human", "{input}")
27     ])
28
29 # Prompt base con instrucciones
30 texto = """Tú eres un asistente para tareas de respuesta a preguntas, haces resúmenes, síntesis,
31 textos, ensayos y puedes ampliar y detallar la respuesta si te lo piden.
32 Usa los siguientes fragmentos de contexto recuperado para responder la pregunta.
33 Si no sabes la respuesta, di que no sabes.
34 Usa un máximo de tres oraciones y mantén la respuesta concisa."""
35
36 # Función principal para RAG
37 def respuesta(pregunta: str, llm, chroma_db, prompt):
38     retriever = chroma_db.as_retriever()
39     chain = create_stuff_documents_chain(llm, prompt)
40     rag = create_retrieval_chain(retriever, chain)
41     results = rag.invoke({"input": pregunta})
42
43 PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
PS C:\Users\ROG\Documents\ROG BASICO> & "c:/Users/ROG/Documents/ROG BASICO/.venv/Scripts/python.exe" "c:/Users/ROG/Documents/ROG BASICO/Class 4 - Version Gratis.py"
Haz tu pregunta: Que exactitud tienen la arquitectura LeNet5 entrenada desde cero con optimizador rmsprop
Respuesta:
Según los datos proporcionados, la exactitud de la arquitectura LeNet5 entrenada desde cero con optimizador RMSProp es:
* Exactitud en entrenamiento: 0.97 (experimento 21), 0.98 (experimento 22), 0.99 (experimento 23), 1.00 (experimentos 24 y 25)
* Exactitud en validación: 0.94 (experimento 21), 0.93 (experimento 22), 0.95 (experimento 23), 0.95 (experimentos 24 y 25)
* Exactitud en prueba: 0.93 (experimento 21), 0.91 (experimento 22), 0.94 (experimento 23), 0.95 (experimentos 24 y 25)
PS C:\Users\ROG\Documents\ROG BASICO>
```