youtube_analysis_PaLM_custom

December 28, 2023

0.1 YouTube Video Analysis on Vertex AI using PaLM and LangChain

[ ]:
```
!pip install --upgrade --user google-cloud-aiplatform>=1.29.0␣
↪google-cloud-storage langchain pytube youtube-transcript-api chromadb gradio␣
↪pydantic
# You may safely ignore dependency warnings below
```

[1]:
```
# restart kernel after installs so that your environment can access the new␣
↪packages
import IPython

app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

[1]:
```
{'status': 'ok', 'restart': True}
```

[1]:
```
# flist = !ls
```

[1]:
```
# get project ID
PROJECT_ID = ! gcloud config get project
PROJECT_ID = PROJECT_ID[0]
LOCATION = "us-central1" # <-- Enter assigned region here.

# generate an unique id for this session
from datetime import datetime
UID = datetime.now().strftime("%m%d%H%M")
```

[ ]:
```
# init the vertexai package
import vertexai
vertexai.init(project=PROJECT_ID, location=LOCATION)
```

[3]:
```
from langchain.document_loaders import YoutubeLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain.llms import VertexAI
```

0.1.1 Load PALM (bison) LLM model

[14]:
```
llm = VertexAI(
    model_name="text-bison@001",
    max_output_tokens=256,
    temperature=0.1,
    top_p=0.8,
    top_k=40,
    verbose=True,
)
```

[15]:
```
# import langchain.llms as LLMS_lc
# help(LLMS_lc)
# help(VertexAI)
```

0.1.2 Load pretrained text embeddings

[16]:
```
from langchain.embeddings import VertexAIEmbeddings

# Embedding
EMBEDDING_QPM = 100
EMBEDDING_NUM_BATCH =5

embeddings = VertexAIEmbeddings(
    requests_per_minute=EMBEDDING_QPM,
    num_instances_per_batch=EMBEDDING_NUM_BATCH,
)
```

```
Model_name will become a required arg for VertexAIEmbeddings starting from
Feb-01-2024. Currently the default is set to textembedding-gecko@001
```

0.1.3 Load YouTube video Transcripts

[ ]:
```
# def set_url(url):
```

[17]:
```
sample_url = "https://www.youtube.com/watch?v=XX2XpqklUrE"

def load_video_transcripts(url=None):

    if url is None:
        url = sample_url

    loader = YoutubeLoader.from_youtube_url(url, add_video_info=True)
    result = loader.load()

    return result
```

0.1.4 Split transcripts into chunks and create embeddings using pre-trained embeddings

[18]:
```
def get_transcript_embeddings(transcript=''):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1500,␣
 ↪chunk_overlap=0)
    docs = text_splitter.split_documents(transcript)
    print(f"# of documents = {len(docs)}")

    embedding_list = embeddings.embed_documents([doc.page_content for doc in␣
 ↪docs])
    print(f"You have {len(embedding_list)} embeddings")
    print(f"Here's a sample of one: {embedding_list[0][:3]}...")

    return docs, embedding_list
```

0.1.5 Index video-transcript as Document embeddings using a vector database here we are using Chroma DB

[19]:
```
def index_and_retrieve(docs=None, embeddings=None):
    db = Chroma.from_documents(docs, embeddings)
    retriever = db.as_retriever(search_type="similarity", search_kwargs={"k":␣
 ↪2}) # here search_kwargs how many top similar documents to return
    return retriever
```

[20]:
```
# help(db.as_retriever)
```

0.1.6 Initialize a Question answering langchain application with palm llm and context info. (via index to vector db)

- here chain_type = 'stuff' indicates how we need to pass the data into the prompt for the llm. i.e. stuff the context data into prompt.

[21]:
```
def get_qa_agent(retriever=None):
    qa = RetrievalQA.from_chain_type( llm=llm, chain_type="stuff",␣
 ↪retriever=retriever, return_source_documents=True)
    return qa
```

[22]:
```
def sm_ask(qa, question, print_results=True):

  video_subset = qa({"query": question})
  context = video_subset
  prompt = f"""
  Answer the following question in a detailed manner, using information from␣
 ↪the text below. If the answer is not in the text,say I dont know and do not␣
 ↪generate your own response.

  Question:
```

```
  {question}
  Text:
  {context}

  Question:
  {question}

  Answer:
  """
  parameters = {
      "temperature": 0.1,
      "max_output_tokens": 256,
      "top_p": 0.8,
      "top_k": 40
  }
  response = llm.predict(prompt, **parameters)
  return {
      "answer": response
  }
```

0.1.7 Using gradio we generate a simple UI for our Question Answering application.

[23]:
```
import gradio as gr
def get_response(url,input_text):

    # load video transcripts
    transcript = load_video_transcripts(url)
    # get embeddings for transcripts
    docs, embedding_list = get_transcript_embeddings(transcript)
    # index embeddings (context data) in vector db
    retriever = index_and_retrieve(docs, embeddings)
    # get qa agent
    qa = get_qa_agent(retriever)

    response = sm_ask(qa, input_text)

    return response

grapp = gr.Interface(fn=get_response, inputs=["text","text"], outputs="text")
grapp.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7864
Running on public URL: https://3ffcb1dc50729e3276.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from Terminal to deploy to Spaces
(https://huggingface.co/spaces)
```

```
<IPython.core.display.HTML object>
```

[23]:

```
# of documents = 9
You have 9 embeddings
Here's a sample of one: [-0.01854412443935871, -0.05032544955611229,
-0.017805462703108788]…
```

0.1.8 sample questions we can use to test the above:

- What is this video about and who are the speakers?
- How is the tone and sentiment of this video?
- Who are the target audience for this video?
- Is this a marketing video by Google Cloud?

[ ]: