Data Science

Assignment no 2

22F-3715

BSE-6B   
  
WEBPAGE SCRAPPING

IN Python

import os

import time

import shutil

import fitz  # PyMuPDF for PDF text extraction

import google.generativeai as genai  # Google Gemini API

import pandas as pd

# 🏢 Define base directory for PDFs (Must be a folder, not a single file)

BASE\_PDF\_PATH = r"C:\Users\Iman Fatima\Desktop\Python\scraper.pdf"  # Update to the correct folder path

OUTPUT\_DIR = r"C:\Users\Iman Fatima\Desktop\Python\paper\_data.csv"  # Base directory for category-wise CSV files

# 🔑 Google Gemini API Key (Replace with your actual API key)

GEMINI\_API\_KEY = "AIzaSyACR4sGAjNECzoc4MopghV5TyvaLyieUGs"

if not GEMINI\_API\_KEY:

    print("❌ Google Gemini API Key is missing! Please update GEMINI\_API\_KEY.")

    exit(1)

# ✅ Configure Google Gemini API

genai.configure(api\_key=GEMINI\_API\_KEY)

# 🎯 Define Five Research Categories

CATEGORIES = [

    "Deep Learning",

    "Computer Vision",

    "Natural Language Processing",

    "Reinforcement Learning",

    "Optimization"

]

CATEGORY\_FILES = {category: os.path.join(OUTPUT\_DIR, f"{category}.csv") for category in CATEGORIES}

def find\_pdfs\_in\_directory(directory):

    """Recursively finds all PDF files in the given directory and its subdirectories."""

    pdf\_files = []

    for root, \_, files in os.walk(directory):

        for file in files:

            if file.endswith(".pdf"):

                pdf\_files.append(os.path.join(root, file))

    return pdf\_files

def extract\_text\_from\_pdf(pdf\_path):

    """Extracts title and abstract from the first page of a PDF."""

    try:

        doc = fitz.open(pdf\_path)

        if len(doc) == 0:

            print(f"⚠ {pdf\_path} is empty or unreadable.")

            return None, None

        text = doc[0].get\_text("text")  # Extract first page text

        doc.close()

        if not text.strip():

            print(f"⚠ {pdf\_path} has no readable text.")

            return None, None

        title = os.path.basename(pdf\_path).replace(".pdf", "").replace("\_", " ")

        abstract = text[:500]  # Extract first 500 characters for classification

        return title, abstract

    except Exception as e:

        print(f"❌ Error extracting text from {pdf\_path}: {e}")

        return None, None

def classify\_paper(title, abstract):

    """Classifies the paper using Google Gemini API with retry mechanism."""

    prompt = f"""Classify the following research paper into one of these categories: {', '.join(CATEGORIES)}.

Title: {title}

Abstract: {abstract}

Category:"""

    model = genai.GenerativeModel("gemini-pro")

    for attempt in range(3):  # Retry up to 3 times

        try:

            response = model.generate\_content(prompt)

            time.sleep(2)  # Add delay to avoid hitting API rate limits

            return response.text.strip()

        except Exception as e:

            print(f"⚠ API Error (Attempt {attempt+1}/3): {e}")

            if "quota" in str(e).lower() or "exhausted" in str(e).lower():

                print("🚨 API quota exceeded! Waiting before retrying...")

                time.sleep(30)  # Wait before retrying

            else:

                break

    return "Unknown"

def save\_to\_csv(papers\_metadata):

    """Saves classified papers into category-specific CSV files."""

    try:

        categorized\_data = {category: [] for category in CATEGORIES}

        categorized\_data["Unknown"] = []  # Handle unknown classifications

        for title, abstract, category, filename in papers\_metadata:

            if category in categorized\_data:

                categorized\_data[category].append([title, abstract, category, filename])

            else:

                categorized\_data["Unknown"].append([title, abstract, "Unknown", filename])

        for category, data in categorized\_data.items():

            file\_path = CATEGORY\_FILES.get(category, os.path.join(OUTPUT\_DIR, "Unknown.csv"))

            df = pd.DataFrame(data, columns=["Title", "Abstract", "Category", "PDF File"])

            if os.path.exists(file\_path):

                df.to\_csv(file\_path, mode='a', header=False, index=False)

            else:

                df.to\_csv(file\_path, index=False)

            print(f"📄 Data saved in: {file\_path} (Total: {len(df)})")

    except PermissionError:

        print("❌ Permission error: Cannot write to CSV files. Close any open files and try again.")

    except Exception as e:

        print(f"❌ Error saving category-wise CSV files: {e}")

def process\_papers(batch\_size=5, daily\_limit=50):

    """Processes PDFs in smaller batches to avoid exceeding API limits."""

    if not os.path.exists(BASE\_PDF\_PATH):

        print(f"❌ Directory not found: {BASE\_PDF\_PATH}")

        return

    print(f"🔍 Scanning for PDFs in {BASE\_PDF\_PATH} and its subfolders...")

    pdf\_files = find\_pdfs\_in\_directory(BASE\_PDF\_PATH)

    if not pdf\_files:

        print("⚠ No PDF files found. Please check the folder structure.")

        return

    print(f"📂 Found {len(pdf\_files)} PDFs")

    papers\_metadata = []

    processed\_count = 0

    for i in range(0, min(len(pdf\_files), daily\_limit), batch\_size):

        batch\_files = pdf\_files[i:i+batch\_size]

        print(f"\n🚀 Processing batch {i//batch\_size + 1} (Files: {len(batch\_files)})...\n")

        for pdf\_path in batch\_files:

            filename = os.path.basename(pdf\_path)

            print(f"📖 Processing: {filename}...")

            title, abstract = extract\_text\_from\_pdf(pdf\_path)

            if title and abstract:

                category = classify\_paper(title, abstract)

                papers\_metadata.append([title, abstract, category, filename])

                print(f"✔ Annotated: {title} → {category}")

                processed\_count += 1

                if processed\_count >= daily\_limit:

                    print("⏸️ Daily API limit reached! Stopping processing.")

                    save\_to\_csv(papers\_metadata)

                    return

        save\_to\_csv(papers\_metadata)

if \_\_name\_\_ == "\_\_main\_\_":

    print("🔎 Starting PDF annotation process...")

    process\_papers(batch\_size=5, daily\_limit=50)  # Process 50 PDFs per day to avoid quota issues

    print("✅ Annotation process completed!")

Link of blog

https://medium.com/@imanfatima7901/automating-research-paper-annotation-with-llms-a-web-scraping-and-ai-powered-approach-1676f6ec2095