# **Zainab Eman**

# **F22-3738**

# **BSSE-6A**

# **DataScience Assignment 01**

## **Scrapper & Annotator script**

import streamlit as st

import requests

from bs4 import BeautifulSoup

import pandas as pd

import aiohttp

import asyncio

import nest\_asyncio

import google.generativeai as genai

# Enable nested asyncio for Streamlit

nest\_asyncio.apply()

# Set up a requests session with connection pooling

session = requests.Session()

adapter = requests.adapters.HTTPAdapter(pool\_connections=100, pool\_maxsize=100)

session.mount('http://', adapter)

session.mount('https://', adapter)

async def fetch\_url(url: str, session: aiohttp.ClientSession, retries: int = 3) -> str:

    """Asynchronously fetch URL content with a custom User-Agent, retry logic, and timeout."""

    headers = {

        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "

                      "AppleWebKit/537.36 (KHTML, like Gecko) "

                      "Chrome/115.0 Safari/537.36"

    }

    for attempt in range(retries):

        try:

            async with session.get(url, headers=headers, timeout=10) as response:

                if response.status == 200:

                    return await response.text()

                else:

                    st.error(f"Error: Received status code {response.status} for {url}")

        except Exception as e:

            if attempt < retries - 1:

                await asyncio.sleep(1)  # Wait before retrying

            else:

                st.error(f"Error fetching {url}: {e}")

                return ""

async def scrape\_paper\_details\_async(paper: dict, session: aiohttp.ClientSession) -> dict:

    """Asynchronously scrape details (abstract) for a single paper."""

    try:

        html = await fetch\_url(paper['link'], session)

        if not html:

            return {'link': paper['link'], 'abstract': "N/A"}

        soup = BeautifulSoup(html, 'html.parser')

        abstract = "N/A"

        abstract\_tag = soup.find('h4', string='Abstract')

        if abstract\_tag:

            abstract\_p = abstract\_tag.find\_next\_sibling('p')

            if abstract\_p:

                abstract = abstract\_p.text.strip()

        return {'link': paper['link'], 'abstract': abstract}

    except Exception as e:

        return {'link': paper['link'], 'abstract': "N/A"}

async def scrape\_papers\_batch(papers: list) -> list:

    """Scrape details for a batch of papers concurrently."""

    async with aiohttp.ClientSession() as session:

        tasks = [scrape\_paper\_details\_async(paper, session) for paper in papers]

        results = await asyncio.gather(\*tasks)

        for paper, details in zip(papers, results):

            paper.update(details)

        return papers

def scrape\_neurips\_page(url: str, year: int, num\_papers: int = None) -> list:

    """

    Scrape papers from a given year's NeurIPS page.

    If num\_papers is None, all papers on the page are returned.

    """

    try:

        response = session.get(url)

        response.raise\_for\_status()

        soup = BeautifulSoup(response.content, 'html.parser')

        papers = []

        paper\_elements = soup.find\_all('li', class\_='conference')

        if num\_papers is not None:

            paper\_elements = paper\_elements[:num\_papers]

        for paper\_element in paper\_elements:

            title\_a = paper\_element.find('a')

            if title\_a:

                title = title\_a.text.strip()

                link = title\_a['href']

                if not link.startswith('http'):

                    link = "https://papers.nips.cc" + link

                authors\_i = paper\_element.find('i')

                authors = authors\_i.text.strip().replace('"', '') if authors\_i else "N/A"

                papers.append({

                    'title': title,

                    'link': link,

                    'authors': authors,

                    'year': year

                })

        return papers

    except Exception as e:

        st.error(f"Error scraping year {year}: {e}")

        return []

def classify\_abstract(abstract: str, api\_key: str) -> str:

    """Use Gemini API to classify the paper abstract."""

    try:

        genai.configure(api\_key=api\_key)

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

        prompt = """

        Classify the following research paper abstract and assign the SINGLE most relevant research area label from this list:

        - Deep Learning

        - Computer Vision

        - Reinforcement Learning

        - NLP

        - Optimization

        Return only the label name, nothing else.

        Abstract:

        {abstract}

        """

        response = model.generate\_content(prompt.format(abstract=abstract))

        return response.text.strip()

    except Exception as e:

        st.error(f"Error classifying abstract: {e}")

        return "Classification Failed"

def main():

    st.title("NeurIPS Papers Scraper and Classifier")

    if st.button("Download papers data"):

        # Years to scrape all papers from

        years = [2022, 2023, 2024]

        all\_papers = []

        base\_url = "https://papers.nips.cc/"

        with st.spinner("Scraping papers from NeurIPS..."):

            try:

                response = session.get(base\_url)

                response.raise\_for\_status()

                main\_soup = BeautifulSoup(response.content, 'html.parser')

            except Exception as e:

                st.error(f"Error accessing NeurIPS main page: {e}")

                return

            # For each year, locate the corresponding link and scrape all papers.

            for year in years:

                year\_link = None

                for link in main\_soup.find\_all('a', href=True):

                    if str(year) in link.text:

                        href = link['href']

                        if not href.startswith("http"):

                            href = base\_url.rstrip("/") + href

                        year\_link = href

                        break

                if not year\_link:

                    st.error(f"Could not find link for year {year}.")

                    continue

                papers = scrape\_neurips\_page(year\_link, year, num\_papers=None)

                if papers:

                    all\_papers.extend(papers)

                else:

                    st.error(f"No papers found for year {year}.")

        if not all\_papers:

            st.error("No papers were scraped.")

            return

        # Asynchronously fetch paper details (e.g., abstracts)

        async def process\_all\_papers():

            return await scrape\_papers\_batch(all\_papers)

        loop = asyncio.new\_event\_loop()

        asyncio.set\_event\_loop(loop)

        processed\_papers = loop.run\_until\_complete(process\_all\_papers())

        df\_scraped = pd.DataFrame(processed\_papers)

        # Replace with your actual Gemini API key

        API\_KEY = "AIzaSyBPuNUYcXYoz7Rwz\_0pvuSFrD4\_sC5R\_5g"

        with st.spinner("Classifying paper abstracts..."):

            labels = []

            for abstract in df\_scraped['abstract']:

                label = classify\_abstract(abstract, API\_KEY)

                labels.append(label)

            df\_scraped['label'] = labels

        st.subheader("Annotated Papers")

        st.dataframe(df\_scraped)

        csv\_data = df\_scraped.to\_csv(index=False)

        st.download\_button(

            label="Download Annotated Data",

            data=csv\_data,

            file\_name="neurips\_2022\_2023\_2024\_annotated.csv",

            mime="text/csv"

        )

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

    main()

## **Git Repo link:** [ZainabEman/DataScience\_Course](https://github.com/ZainabEman/DataScience_Course)

## **Blog link:** [Scraping and Annotating NeurIPS Papers: A Data Science Adventure | by ZainabEman | Feb, 2025 | Medium](https://medium.com/@zainabeman976/scraping-and-annotating-neurips-papers-a-data-science-adventure-dc670c40fae6)