PubMed Article Summarization and Image Generation Web Application

Introduction

This project aims to develop a web application that summarizes PubMed articles and generates images using OpenAI's DALL-E model. The application leverages Streamlit for the web interface, Natural Language Toolkit (NLTK) for text preprocessing, and OpenAI's language models for text summarization and image generation.

Instructions

1. Environment Setup

Ensure you have Python installed. You will also need to install the required libraries.

```
pip install streamlit openai nltk langchain
```

2. Data Exploration

For this project, we use the `ccdv/pubmed-summarization` dataset from the Hugging Face `datasets` library. This dataset contains PubMed articles specifically curated for summarization tasks.

3. Model Selection and Fine-Tuning

We use OpenAI's language models for text summarization. There are two types of summarization methods implemented: "Brief" and "Detailed", which use different chain types in Langchain. For image generation, we use OpenAI's DALL-E model.

4. Web Application Development

We develop the web application using Streamlit. The application has a user-friendly interface where users can input PubMed articles, select the type of summary, and optionally generate images based on the summary.

5. Code Explanation

After careful analysis of the given dataset and exploring PubMed articles from the National Institute of Health website, I concluded that such a large text dataset requires narrowing down.

- Cleaning

```
def clean(txt):
    txt = re.sub(r'[^a-zA-Z0-9\s]','',txt)
```

```
    return txt
```

Here I utilized the `clean` function to preprocess a given text by removing all special characters except for alphanumeric characters (letters and numbers) and whitespace as the PubMed articles usually contains valuable statistical data but also some links and reference Ids which can be omitted.

- Removing Stop words:

```
def remStopWords(txt):
    stoplist = set(stopwords.words('english'))
    word_tokens = word_tokenize(txt)
    filtered_text = [word for word in word_tokens if word.lower() not in
stoplist]
    filtered_text = ' '.join(filtered_text)
    return filtered_text
```

Here I removed all the words which are used in speech but hold less value in the data. I imported the nltk library to download these stop words and filter them from the given text so that a lengthy article containing explanatory jargon can be reduced.

- Lemmatization:

```
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts."""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {
        'J': wordnet.ADJ,
        'N': wordnet.NOUN,
        'V': wordnet.VERB,
        'R': wordnet.ADV
    }
    return tag_dict.get(tag, wordnet.NOUN)

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    word_tokens = word_tokenize(text)
    lemmatized_text = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for
word in word_tokens]
    return ' '.join(lemmatized_text)
```

Here I again used the nltk library to download dependencies and then made use of wordnet to gather the dictionary to eliminate excessive wordings and reduce the adjectives, verbs or nouns to reduce the data and only use the root meanings for better summarization.

- OpenAI Integration:

```
def generate_response(txt):
```

```
    # Instantiate the LLM model
    llm = OpenAI(temperature=0, openai_api_key=api_key)

    # Split text
    text_splitter = CharacterTextSplitter()
    texts = text_splitter.split_text(txt)

    # Create multiple documents
    docs = [Document(page_content=t) for t in texts]

    # Text summarization
    chain = load_summarize_chain(llm, chain_type='map_reduce')
    return chain.run(docs)
```

Here I used LangChain llm model to integrate OpenAI. This basically works by setting a creativity level for the llm and authenticating the api key, afterwards split the long text so that the data can be handled in chunks called documents. This is necessary for the load summarize chain function as some articles may contain longer text which are not suitable for OpenAI sometimes and may cause excessive overload. The above shown version uses map_reduce option to give a brief and deterministic summary and for detailed I made another function in which I increased the creativity temperature and used stuff option to provide a longer summary.

- Image Generation by DALL-E:

```
def generate_image(prompt):
    response = openai.Image.create(
        prompt=prompt,
        n=1,
        size="1024x1024"
    )
    image_url = response['data'][0]['url']
    return image_url
```

This is pretty straight forward. This again uses the same openai library to generate a image and return its url for display.

- Streamlit form and Api key authentication:

```
with st.form('summarize_form', clear_on_submit=True):
    api_key = st.text_input('OpenAI API Key', type='password', disabled=not
text)
    submitted = st.form_submit_button('Submit')
    if submitted and api_key.startswith('sk-'):
        with st.spinner('Calculating...'):
            if option == "Brief":
                response = generate_response(text)
            elif option == "Detailed":
                response = generate_response2(text)
```

```
        summarized.append(response)
        if len(summarized):
            st.info(response)
        if(option2 == "Yes"):
            image_url = generate_image(response)
            response = requests.get(image_url)
            image = Image.open(BytesIO(response.content))
            st.image(image, caption=response)
        del api_key
```

Here I used the exception handling in python to create a form and input the api key which is authenticated and kept hidden and displays an spinner after which the response is generated using above functions according to the check boxes submitted. To ensure safety the key is deleted at the end of the application.

6. Running the Application

To run the Streamlit application, execute the following command in your terminal:

```
streamlit run application.py
```

To get your api key go to OpenAI https://platform.openai.com/account/api-keys and either generate your new secret key or copy your previous keys. Make sure you have enough credits or the application won't work.

7. Conclusion

This application demonstrates a complete pipeline for summarizing PubMed articles and generating images based on the summaries. It covers data preprocessing, model selection, fine-tuning, and web application development using Streamlit. By following the instructions and using the provided code, you can create your own summarization and image generation tool.