News Summarization & Text-to-Speech (BBC Edition)

-- Sayan Das

I have built a two-part application that fetches BBC news articles for a given company, performs summarization and sentiment analysis on those articles, and finally provides a Hindi text-to-speech (TTS) summary. The frontend is written in **Streamlit** and is hosted on **Hugging Face Spaces**, while the backend is a **Flask** API hosted on **AWS Lightsail**. Below is a comprehensive guide to help you understand how I set up the project, the models I used, how the API is structured, and any assumptions or limitations.

---

1. Demonstration

For a quick overview, please watch the demonstration here:
Video - https://youtu.be/9FTq97tuEOk

Live website - https://huggingface.co/spaces/Nobita69/News-Summarization

Project Overview

This application extracts news articles related to a given company, performs sentiment analysis, conducts a comparative sentiment analysis, and generates a Hindi TTS audio summary.

Project Structure (Imp files)

```
Akaike_Technologies-Data_Science/
├── app.py          # Streamlit frontend
├── api.py          # Flask backend API
├── utils.py        # Utility functions for news scraping, summarization, sentiment
analysis, and TTS
├── requirements.txt
└── README.md
```

2. Project Setup

2.1. Installation and Environment

1. **Clone or download** this repository onto your local machine.
2. **Install Dependencies:**

```
py --list
py -3.10 -m venv env
env\Scripts\activate
python --version

pip install -r requirements.txt
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
python -m spacy download en_core_web_md

(To run on CUDA 12)
nvcc --version
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
```

3. **Run the Backend Flask App:**

```
python api.py
```

The API will start on `http://localhost:5000`.

4. **Run the Frontend Streamlit Application:** In another terminal, run:

```
streamlit run app.py
```

5. **Usage:**
   - Enter a company name in the sidebar (e.g., Tesla, Meta, Apple).
   - Click Analyze to fetch the latest BBC articles and run sentiment analysis.
   - View the results (final sentiment, chart, extended analysis, articles), plus an audio summary in Hindi.
   - After analyzing, a Search Query (Semantic) section appears in the sidebar. Enter a query to find relevant articles.
   - If you want to return to the original analysis view, click Back to Full Analysis.

2.2. Production Hosting

- **Backend**: Deployed on **AWS Lightsail** with Gunicorn (or another WSGI server) to ensure 24/7 availability.

- **Frontend**: Hosted on **Hugging Face Spaces**, so users can access the Streamlit interface online.

---

3. Model Details

I rely on several NLP and speech models to handle summarization, sentiment analysis, and TTS:

1. **Summarization**
   - **Hugging Face Transformers** (`sshleifer/distilbart-cnn-12-6`):
     I use a pre-trained DistilBART model to summarize each article's content if it exceeds a certain length. If the text is short, I fall back to a simpler sentence-based summarization using NLTK's `sent_tokenize`.
2. **Sentiment Analysis**
   - **DistilBERT** (`distilbert-base-uncased-finetuned-sst-2-english`):
     This model provides a binary sentiment classification (Positive/Negative). If this pipeline fails, I fall back to **VADER** (`vaderSentiment`) for a sentiment score.
3. **Topic Extraction**
   - **KeyBERT**:
     KeyBERT extracts the top keywords or key phrases from the summarized text. If KeyBERT fails or isn't available, I use **spaCy**'s noun-chunks to find frequently mentioned terms.
4. **TTS (Text-to-Speech)**
   - **gTTS (Google Text-to-Speech)**:
     After I generate the final sentiment summary, I translate the text to Hindi using **googletrans** and then create an MP3 file using gTTS. The audio file is temporarily stored on the server and removed once it has been served to the client.
5. **Semantic Search**
   - **SentenceTransformer** (`all-MiniLM-L6-v2`):
     Each article's summary is converted into embeddings. Users can perform a semantic search

by entering a query, which is also converted into an embedding. I then compute similarity scores between the query and each article to show the most relevant matches.

---

4. API Development

4.1. Flask App Structure

- **`api.py`**:
  - Initializes the Flask application.
  - Defines the `/analyze` endpoint for handling requests with a JSON body like `{"company": "Tesla"}`.
  - Uses functions from `utils.py` to fetch, process, and analyze BBC articles.
  - Returns a JSON response containing:
    - Basic article info (title, summary, sentiment, topics).
    - Comparative and extended analysis results.
    - Sentence embeddings for semantic search.
    - The name of the generated TTS MP3 file.
  - Defines the `/audio/<filename>` endpoint to serve the generated MP3 file and then delete it afterward.
- **`utils.py`**:
  - Handles all the logic for fetching news from BBC, summarizing text, analyzing sentiment, extracting topics, building embeddings, and generating TTS audio.
  - Uses **asyncio** and **aiohttp** to fetch article contents concurrently.
  - Maintains a simple in-memory cache for the fetched articles to reduce redundant network calls.
  - Translates text to Hindi using `googletrans` and then creates MP3 files with `gTTS`.

4.2. Endpoints

1. **`GET /`**
   - Returns a simple JSON message: `{"status": "Backend is running!"}`.
   - Used to check if the backend is up.
2. **`POST /analyze`**
   - Expects a JSON body like:

     ```
     {
       "company": "Tesla"
     }
     ```

   - Fetches up to 15 BBC articles for the given company, processes them, and returns the analysis in JSON format.

   - Example of a JSON response (truncated for brevity):

     ```
     {
       "Company": "Tesla",
       "Articles": [...],
       "Comparative Sentiment Score": {...},
       "Extended Analysis": {...},
       "Final Sentiment Analysis": [...]",
       "Audio": "tts_<some_id>.mp3"
     }
     ```

3. **`GET /audio/<filename>`**
   - Serves the generated MP3 file, then deletes it from the server after sending it.
   - If the file is not found, returns a 404 JSON error.

---

5. API Usage

5.1. Consuming the API

You can interact with the backend via tools like **Postman** or **cURL**:

- **POST /analyze** (JSON Body)
```
curl -X POST \
     -H "Content-Type: application/json" \
     -d '{"company":"Ford"}' \
     http://<server_ip_or_domain>:5000/analyze
```

- **GET /audio/**
```
curl http://<server_ip_or_domain>:5000/audio/tts_xxxx.mp3 --output local_audio.mp3
```

5.2. Third-Party APIs

- **BBC Search**:
  I am scraping BBC's search pages (publicly available) to retrieve articles for a given company name.
- **Googletrans**:
  Used for language translation to Hindi.
- **gTTS**:
  Used to generate Hindi audio.

I integrated these services so I can automatically fetch news content, analyze it, translate the final summary, and create audio output.

---

6. Assumptions & Limitations

1. **BBC Search Availability**:
   I assume BBC's search pages remain publicly accessible and do not change their HTML structure significantly. If BBC updates its layout, my scraping logic may need adjustments.

2. **Internet Connectivity**:
   The application relies on external services (BBC, googletrans, gTTS). A stable internet connection is assumed for fetching news and generating TTS audio.

3. **Limited Article Summaries**:
   I only retrieve and summarize the first three paragraphs of each BBC article. This may not capture the full context of the article.

4. **Hindi TTS Accuracy**:
   The translation and TTS rely on Google's services. The quality of the translation and speech synthesis may vary.

5. **Caching**:
   I use an in-memory cache for a few minutes to avoid refetching the same articles repeatedly. This

means the server's memory usage grows with more requests, and it resets when the server restarts.

6. **No User Authentication**:
   The application is open. There is no login or rate-limiting mechanism in place.

7. There's enough resource available - otherwise it will give [ERROR] Worker (pid:xx) was sent SIGSEGV!

7. Conclusion

This project demonstrates how I can integrate multiple NLP pipelines—news fetching, summarization, sentiment analysis, topic extraction, and TTS—into a cohesive application. The frontend in Streamlit (hosted on Hugging Face Spaces) interacts seamlessly with the Flask backend (hosted on AWS Lightsail). Users can search for a company, read summarized articles, see sentiment and topic analyses, and even listen to a Hindi audio summary.