Internship Project at IntelliInvest: Real-Time Financial News Tracking and Summarization

Kushal C. Gajbe
B.Tech Electrical Engineering, IIT Bombay

Introduction

During my internship at IntelliInvest, I helped build a smart system for tracking and summarizing financial news in real-time. Using Python, I connected different software tools to gather and analyze data from websites like DuckDuckGo and Yahoo Finance. The system automatically collected news, followed stock market trends like Nifty50 and Sensex, and predicted market feelings before and after trading. I used advanced computer techniques to write clear summaries without personal opinions, tailored for traders. This project automated how we share market updates with clients instantly.

Explanation of Python Code

The Python code performs the following tasks:

1. **RSS Feed URLs**: Defines a list `rss_url` containing URLs of RSS feeds from which news articles will be fetched.
2. **Time Handling**:
   - Retrieves the current date and time in the IST timezone (`Asia/Kolkata`) using `datetime.now()`.
   - Calculates `yesterday_ist` by subtracting one day from `now_ist`.
3. **Interval Definition Functions**:
   - `define_summary_intervals(now_ist)`: Defines intervals (`pre_market_1`, `pre_market_2`, etc.) during which summaries will be generated based on the current time.
   - `define_run_intervals(now_ist)`: Defines intervals (`detect_pre_market_1`, `detect_market_hours_1`, etc.) during which the code will run to detect when to summarize news based on market hours.
4. **Interval Determination**:
   - `get_summary_interval(now_ist, run_intervals, summary_intervals)`: Determines the specific interval (`summary_start` to `summary_end`) and the file name (`file_to_save`) for summarization based on the current time and predefined run intervals.
5. **RSS Feed Parsing**:
   - `extract_links(rss_url, now_ist, summary_start, summary_end)`: Parses RSS feeds (`rss_url`) to extract links of articles published within the specified `summary_start` and `summary_end` times.
6. **Using DuckDuckGo Search for Pre-Market**:
   - **Time Check**:
     - Checks if the current time is between 8:20 AM and 8:30 AM IST.
   - **Search Execution**:
     - Initiates a search for news related to "Nifty50" using DuckDuckGo.
     - Uses the DuckDuckGo Search API wrapper (`DuckDuckGoSearchAPIWrapper`) configured for the region "in-en" and time filter "d" (day) to fetch up to 4 results.
   - **Result Extraction**:

    - Extracts relevant fields (title, link, date) from the search results using regular expressions (`re.compile`).
    - Stores the extracted results in a list of dictionaries (`results`), where each dictionary represents a news article with keys for title, link, and date.

7. **Article Content Extraction**:
    - Defines a function `extract_article_content(url)` that:
        - Uses `requests.get(url)` to fetch the HTML content from a given `url`.
        - Parses the HTML content using BeautifulSoup (`BeautifulSoup(response.text, 'html.parser')`).
        - Extracts the title from the HTML ‘ $< title >$ ‘ tag if present; otherwise, assigns ”No title found”.
        - Retrieves text content from all ‘ $< h1 >$ ‘, ‘ $< h2 >$ ‘, and ‘ $< p >$ ‘ tags, combining headers and paragraphs into a single string (`content`).
        - Returns a dictionary with keys for ’title’ and ’content’, containing the extracted title and combined text content, respectively.

8. **Text Cleaning Function**:
    - Defines a function `clean_text(content)` that:
        - Uses regular expressions (`re.sub`) to remove unwanted characters (excluding alphanumeric, whitespace, comma, period, exclamation mark, question mark, single quote, double quote, and hyphen).
        - Replaces multiple whitespace characters with a single space.
        - Strips leading and trailing whitespace from the cleaned content.
        - Returns the cleaned text content.

9. **Prompt Generation**:
    - Defines two prompts (`prompt1` and `prompt2`) for requesting summaries from a model (`ollama`). Each prompt specifies requirements for summarizing financial market information, including support and resistance levels or closing levels of indices like Nifty50, Bank Nifty, and Sensex.

10. **Ollama Model Interaction**:
    - Initializes an instance of the Ollama model (`ollama`) configured to connect to a local server (`base_url='http://localhost:11434'`) and using the ’llama3’ model variant.
    - Defines a function `query_model1(prompt1, prompts)` that:
        - Combines multiple prompts into a single input string and calls the Ollama model with the combined prompt to generate a summarized response.
        - Measures and prints the time taken for the model to process the batch of prompts.
        - Returns the model’s response.

11. **Ticker Data Retrieval**:
    - Defines a dictionary (`ticker_dic`) mapping ticker names to their corresponding symbols for financial indices like Nifty50, Bank Nifty, and Sensex.
    - Defines a function `fetch_ticker_data(ticker_symbol)` that:
        - Uses the Yahoo Finance API (`yf.Ticker(ticker_symbol)`) to fetch historical data for the past 5 days for a given ticker symbol.
        - Retrieves today’s and yesterday’s closing prices, high and low prices, open price, percentage change, and price difference for the specified ticker symbol.
        - Returns a dictionary containing the fetched data.

12. **DuckDuckGo for Post-Market News and Data Fetching**:
    - Checks if the current time is between 4:30 PM and 4:40 PM IST.
    - If true, fetches news related to Nifty50 using DuckDuckGo for the post-market summary.
        - Initializes a DuckDuckGoSearchAPIWrapper (`wrapper`) configured for news search in India with a maximum of 4 results.
        - Executes a search for ”Nifty50” and extracts titles, links, and dates using regular expressions (`re.compile`).

  - Stores results in a list of dictionaries (`results`) containing title, link, and date for each news item.
  - Filters and prints news items published between 12:00 PM and the summary end time (post-market summary interval).
- Prints completion messages after fetching news related to Nifty50 and proceeds to fetch data from Yahoo Finance for Nifty50, Bank Nifty, and Sensex.
- Stores today's financial market main indices closing values and changes from the previous close in `todays_headlines`.
  - Formats the closing values and changes for Nifty50, Bank Nifty, and Sensex, including percentage changes and point differences.
  - Writes `todays_headlines` into a text file named `market_index_closing.txt`.

13. **Summarizing Content from Selected Links**:
    - Initializes an empty string `final_summary` to store the summarized content.
    - Defines a function `get_batch_size(t)` to determine the batch size based on the current hour (`now_ist.hour`). Adjusts batch size dynamically between 2 and 3 based on the hour of execution.
    - Iterates through `links` in batches based on `batch_size`.
      - Cleans and extracts article content for each link using `extract_article_content` and `clean_text`.
      - Queries the model (`ollama`) with the cleaned article content as prompts, using either `prompt1` or `prompt2` based on the hour.
      - Appends the obtained summaries to `final_summary`.
    - Prints completion messages after summarizing all batches.
    - Saves `final_summary` to a text file named `{file_to_open}.txt`.
    - If the current time is between 8:15 AM and 9:00 AM or between 4:30 PM and 5:10 PM, executes `all_final.py` and `clean_send.py` sequentially.

Explanation of all_final.py

1. **Setting Up Script**:
   - Retrieves the current date and time in Indian Standard Time (IST) using `datetime.now(pytz.timezone('Asia/Kolkata'))`.
   - Defines a function `read_summary(file_path)` to read contents from specified text files (`pre_sum1.txt`, `pre_sum2.txt`, etc.).
   - Reads summaries from various text files into variables `pre_summary1`, `pre_summary2`, `post_summary1`, `post_summary2`, `post_summary3`, `post_summary4`, and `market_headlines`.
2. **Determining Summary Type**:
   - Compares the current hour (`now_ist.hour`) to decide whether to use pre-market or post-market summaries. Updates `all_summaries` and sets `word` accordingly.
   - Prints the determined summary type.
3. **Setting Prompts for Model Query**:
   - Defines prompts `prompt_post` and `prompt_pre` for querying the model based on whether it's pre-market or post-market.
4. **Querying the Model**:
   - Initializes an instance of the Ollama model (`ollama`) configured to a specific base URL and model type.
   - Calls the `query_model` function to query the model with the appropriate prompt and all collected summaries (`all_summaries`).
   - Saves the final model output to a text file named `final_output.txt`.

Explanation of clean_send.py

1. **Imports and Setup**:
    - Imports necessary libraries such as `requests`, `BeautifulSoup`, `MongoClient`, `datetime`, `pytz`, `nltk`, `feedparser`, `re`, `time`, `json`, and modules from `langchain` and `langchain_community`.
    - Initializes the current date and time in Indian Standard Time (IST) using `datetime.now(pytz.timezone('Asia/Kolkata'))`.
    - Defines a function `query_model(prompt)` to query the Ollama model for text summarization.
    - Initializes an instance of the Ollama model (`ollama`) with specific configuration (`base_url` and `model`).
2. **Reading Final Output**:
    - Reads the contents of `final_output.txt` which contains the summarized market information.
    - Removes the first two lines from the read content to exclude model information and non-content lines.
3. **Setting Up Prompt and Querying Model**:
    - Defines `prompt` to instruct the model to remove specific lines from the output.
    - Queries the Ollama model with the combined prompt and cleaned final output to get a refined summary.
4. **Cleaning Response**:
    - Removes '**' characters from the response obtained from the model to clean up formatting.
    - Writes the cleaned response to `clean_output.txt`.
5. **Preparing Data for API Call**:
    - Reads the cleaned response from `clean_output.txt`.
    - Gets the current time in ISO 8601 format with UTC.
    - Determines the title (`"Pre-Market Summary"` or `"Post-Market Summary"`) based on the current time.
    - Constructs a dictionary `data` containing title, article content, published time, and image information for the API call.
6. **Sending Data via API**:
    - Defines the API endpoint URL (`url`) and headers (`headers`).
    - Sends a POST request to the API endpoint with JSON data containing the prepared `data`.
    - Checks the response status code and prints success or failure messages accordingly.