By Langat Evans

Sentiment Analysis API with Groq Integration

1. Introduction

This project aimed to create a Python-based API that allows users to upload CSV or XLSX files containing customer reviews. The API processes these reviews, performs sentiment analysis using the Groq API, and returns structured sentiment scores in JSON format.

2. Project Requirements

The project required the following functionalities:

- **API Development**: Build a Flask-based API (using Django instead) that accepts CSV/XLSX files with customer reviews.
- **Sentiment Analysis**: Integrate the Groq API to classify reviews into positive, negative, or neutral categories.
- **Error Handling**: Implement error handling for invalid files or API issues.
- **Documentation**: Provide a PDF detailing the approach, findings, and API examples.

3. Development Process

API Design

The project was implemented using the **Django** framework instead of Flask. The application consists of the following key components:

- **File Upload Endpoint (`/upload/`)**: This endpoint accepts a file in CSV or XLSX format, processes the reviews, and sends them for sentiment analysis.
- **Sentiment Analysis Endpoint (`/sentiment/`)**: This is where the actual sentiment analysis happens. It reads the uploaded file, extracts the review text, and sends it to the Groq API.

The structure of the Django application is as follows:

- `views.py`: Handles file uploads, parsing, and API calls to Groq.
- `urls.py`: Defines URL routing for the API endpoints.
- `templates/`: Contains the HTML template for the file upload form.
- `settings.py`: Configures Django settings for the project.

File Handling (CSV/XLSX Parsing)

The API can handle both CSV and XLSX file formats using the `Pandas` library. The uploaded file is parsed to extract customer reviews from a column labeled `Review`. If this column is missing or the file is improperly formatted, the API returns an appropriate error message.

The following steps are performed during file handling:

- **Upload**: A user uploads a CSV/XLSX file containing customer reviews.
- **Parsing**: The file is parsed using `Pandas` to extract text data.
- **Validation**: The extracted data is validated to ensure the `Review` column exists.

Groq API Integration

The core functionality of the project involves sending customer reviews to the **Groq Sentiment Analysis API** and processing the response. Here's how this integration works:

- **API Request**: For each review, an HTTP POST request is made to the Groq API.
- **Response Parsing**: The response from Groq contains sentiment scores in positive, negative, and neutral categories.
- **Aggregation**: The API aggregates the sentiment scores from all reviews and returns a final JSON response.

The Groq API request and response handling is implemented as follows:

```
import requests
```

```
def perform_sentiment_analysis(reviews):
    api_url = "https://api.groq.com/sentiment"
    headers = {"Authorization": "Bearer YOUR_GROQ_API_KEY"}

    positive_score, negative_score, neutral_score = 0, 0, 0

    for review in reviews:
        try:
            response = requests.post(api_url, headers=headers, json={"text": 
review})
            response.raise_for_status()
            sentiment = response.json()
            positive_score += sentiment.get('positive', 0)
            negative_score += sentiment.get('negative', 0)
            neutral_score += sentiment.get('neutral', 0)

        except requests.exceptions.RequestException as e:
            return {"error": str(e)}

    return {
        "positive": positive_score,
        "negative": negative_score,
        "neutral": neutral_score
    }
```

Error Handling

The API implements basic error handling for the following cases:

- **Invalid File Format**: If the uploaded file is not CSV or XLSX, an error message is returned.
- **Missing `Review` Column**: If the `Review` column is missing, the API returns a descriptive error message.
- **Groq API Errors**: If the Groq API returns an error (e.g. network issues or authentication failure), a corresponding error is passed back to the user.

4. Example API Usage

Request and Response

Request:

```
POST http://127.0.0.1:8000/sentiment/
Content-Type: multipart/form-data

file: customer_reviews.csv
```

Example CSV (`customer_reviews.csv`):

```
Review
"The product is amazing!"
"Terrible experience, would not recommend."
"Service was okay, but could be better."
```

Response:

```
{
    "positive": 1,
    "negative": 1,
    "neutral": 1
}
```

Sample Input/Output

- **Input**: A file containing 3 customer reviews, one positive, one negative, and one neutral.
- **Output**: A JSON object indicating the count of positive, negative, and neutral reviews.

5. Findings and Analysis

Results of Sentiment Analysis

After testing the API with a sample dataset of customer reviews, the results were consistent with expectations. Positive reviews were correctly classified, and negative reviews were accurately flagged. Neutral reviews were also identified when the sentiment was mixed or ambiguous.

Performance and Limitations

1. **Performance**: The API performed reasonably well for a small number of reviews. However, processing large datasets could introduce delays due to the sequential nature of API calls to Groq.
2. **Rate Limiting**: Depending on the Groq API's rate limits, processing too many reviews in a short time may cause the API to reject requests. This issue could be mitigated by adding rate limiting or batching requests.
3. **Granularity**: The API only returns broad sentiment categories (positive, negative, neutral). More granular sentiment scoring (e.g. very positive, somewhat negative) could be implemented to improve the insights provided by the system.

Improvements and Future Work

- **Batch Processing**: To improve performance, reviews could be sent in batches rather than individually if the Groq API supports bulk requests.
- **Advanced Error Handling**: More sophisticated error handling could be added to manage edge cases, such as API downtime or invalid responses from Groq.
- **Custom Sentiment Model**: Future iterations could integrate a custom-trained sentiment model to handle domain-specific customer feedback.

6. Conclusion

This project successfully implements a Django-based API for sentiment analysis using the Groq API. The API accepts CSV/XLSX files, processes customer reviews, and returns structured sentiment scores. While the system performs well for smaller datasets, improvements can be made in terms of handling larger volumes of data and introducing more granular sentiment categories.

Thanks for reading the documentation.

Welcome again...