Sentiment Analysis API with LLM Integration

Author: Aravind Reddy Gudi
Date: September 2024

1. Approach to Solving the Problem

The goal is to implement an API that allows users to upload a CSV or XLSX file containing customer reviews, and then return sentiment analysis results (positive, negative, neutral) for each review using the Groq API for enhanced LLM-based analysis.

Key Considerations:

- - File Handling: The API should support common file types like CSV and XLSX, handling potential errors like missing columns or incorrect formats.
- - Sentiment Analysis: Using a Large Language Model (LLM) through the Groq API to classify reviews as positive, negative, or neutral.
- - Structured Response: The API must return results in a clear and structured JSON format, allowing for easy integration into other systems.
- - Error Handling: Robust error management is crucial to handle potential issues with file upload or processing.

2. How the Structured Response Was Implemented

The structured response is implemented using Flask's jsonify function, which returns a JSON object after analyzing the sentiment of uploaded reviews. The response is organized to include a count of positive, negative, and neutral reviews, ensuring clarity and ease of interpretation.

Implementation:

```
return jsonify({
   "positive": sentiment_counts["positive"],
   "negative": sentiment_counts["negative"],
   "neutral": sentiment_counts["neutral"]
})
```

Example Output:

```
{
   "positive": 5,
   "negative": 3,
   "neutral": 2
}
```

3. Examples of API Usage with Sample Inputs/Outputs

Example 1: Positive Sentiment Review

Input Review: "This product exceeded my expectations!"

Sentiment Analysis: Positive

Output:

```
{
  "positive": 1,
  "negative": 0,
  "neutral": 0
}
```

Example 2: Negative Sentiment Review

Input Review: "The product broke within a week, very disappointed."

Sentiment Analysis: Negative

Output:

```
{
  "positive": 0,
  "negative": 1,
  "neutral": 0
}
```

Example 3: Neutral Sentiment Review

Input Review: "It does what it's supposed to do, nothing more, nothing less."

Sentiment Analysis: Neutral

Output:

```
{
  "positive": 0,
  "negative": 0,
  "neutral": 1
}
```

4. Analysis of Results, Including Limitations and Potential Improvements

Strengths:

- - Ease of Use: The API simplifies sentiment analysis by allowing users to upload a file and quickly obtain results.
- - Structured Output: JSON responses are clear, making it easy to integrate into other applications.
- - Error Handling: The API checks for file types and column presence, ensuring robustness.

Limitations:

- - Basic Sentiment Categories: The sentiment classification is limited to positive, negative, and neutral. More nuanced sentiments, like "mixed" or "sarcastic," aren't captured.
- - Keyword Bias: In cases where simple keyword-based analysis is used, it might misclassify sentiment, as it does not account for context.
- - LLM Limitation: The Groq API integration provides better context analysis, but performance may vary depending on the model used.

Potential Improvements:

- - Advanced Sentiment Categories: Incorporating more sentiment categories (e.g., very positive, very negative) can enhance the granularity of analysis.
- - Multilingual Support: Expanding support to handle reviews in multiple languages would make the API more versatile.
- - Batch Processing Optimization: For larger datasets, the API could benefit from optimizations in the LLM processing to reduce latency.
- - Customization: Allowing users to set certain parameters, such as sensitivity to negative or positive sentiment, could make the tool more flexible.

5. Additional Insights and Observations

Observations:

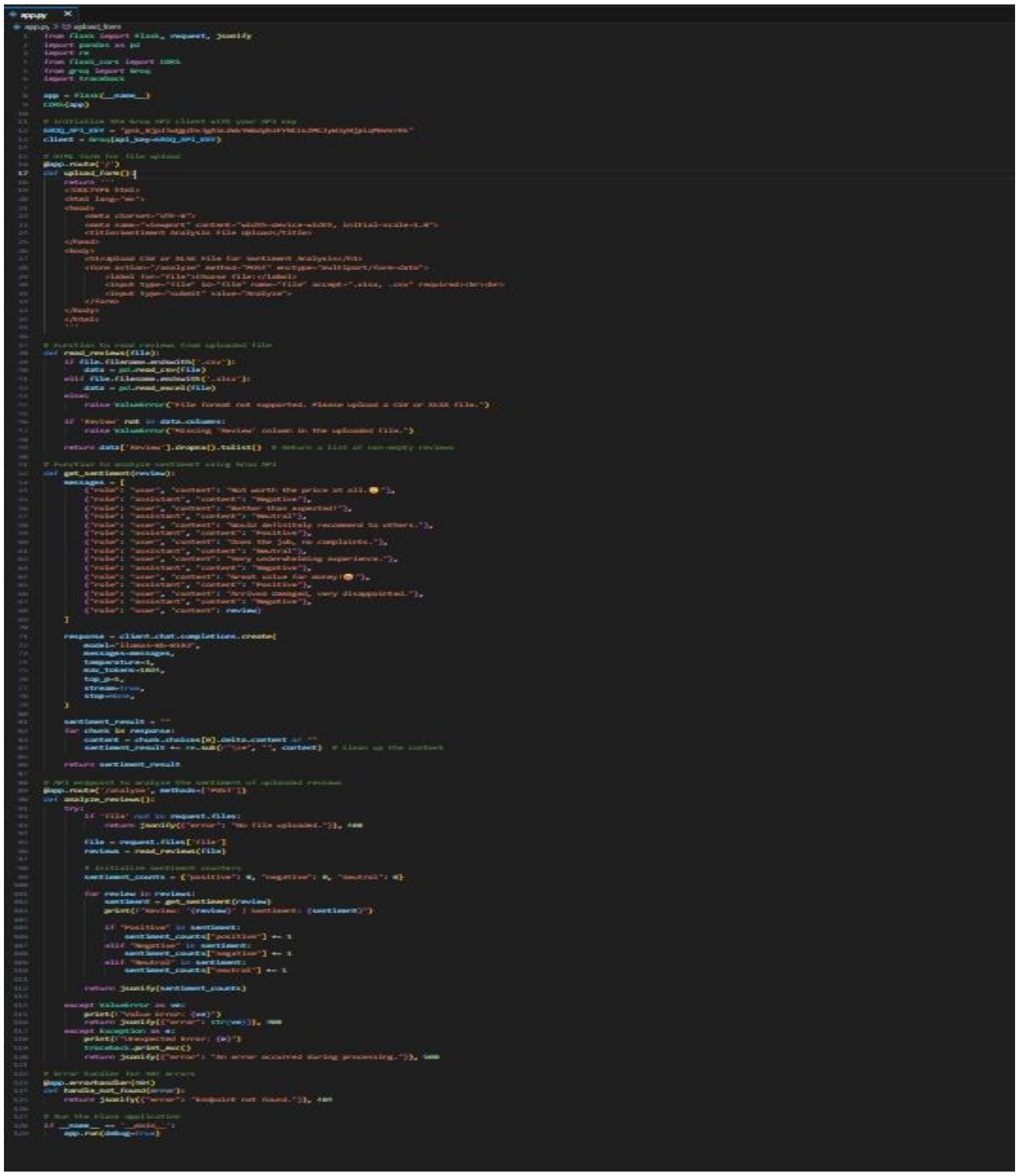
- - Handling Emojis and Special Characters: The Groq API manages emojis and symbols well, which often add context to sentiment (e.g., " 😃 " indicating positivity). It’s important to ensure these are processed correctly.

- - Flexibility: By utilizing LLMs, this solution moves beyond keyword-based analysis, providing more accurate and context-sensitive classification.

Insights:

- - User Experience: The HTML form interface simplifies file uploads, making the API accessible even to users with minimal technical expertise. For more advanced users, the API endpoint can be easily integrated into other systems.
- - Scalability: With improvements, this API could be scaled to handle larger datasets or integrated into business analytics platforms for processing customer feedback in real-time.

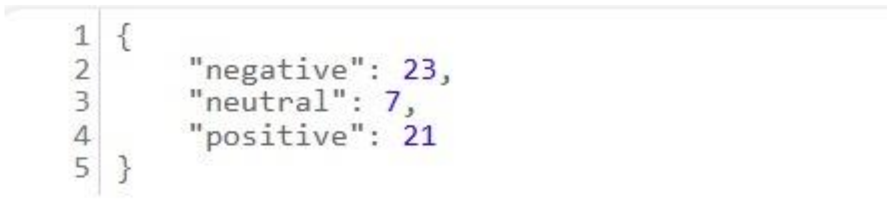
Code:


app.py ×

app.py > upload_form

```python
from flask import Flask, request, jsonify
import pandas as pd
import re
from flask_cors import CORS
from groq import Groq
import traceback

app = Flask(__name__)
CORS(app)

# Initialize the Groq API client with your API key
GROQ_API_KEY = "gsk_[illegible]"
client = Groq(api_key=GROQ_API_KEY)

# HTML form for file upload
@app.route('/')
def upload_form():
    return '''
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Sentiment Analysis File Upload</title>
    </head>
    <body>
        <h1>Upload CSV or XLSX File for Sentiment Analysis</h1>
        <form action="/analyze" method="POST" enctype="multipart/form-data">
            <label for="file">Choose file:</label>
            <input type="file" id="file" name="file" accept=".xlsx, .csv" required><br><br>
            <input type="submit" value="Analyze">
        </form>
    </body>
    </html>
    '''

# Function to read reviews from uploaded file
def read_reviews(file):
    if file.filename.endswith('.csv'):
        data = pd.read_csv(file)
    elif file.filename.endswith('.xlsx'):
        data = pd.read_excel(file)
    else:
        raise ValueError("File format not supported. Please upload a CSV or XLSX file.")

    if 'Review' not in data.columns:
        raise ValueError("Missing 'Review' column in the uploaded file.")

    return data['Review'].dropna().tolist()  # Return a list of non-empty reviews

# Function to analyze sentiment using Groq API
def get_sentiment(review):
    messages = [
        {"role": "user", "content": "Not worth the price at all.[illegible]"},
        {"role": "assistant", "content": "Negative"},
        {"role": "user", "content": "Better than expected!"},
        {"role": "assistant", "content": "Neutral"},
        {"role": "user", "content": "Would definitely recommend to others."},
        {"role": "assistant", "content": "Positive"},
        {"role": "user", "content": "Does the job, no complaints."},
        {"role": "assistant", "content": "Neutral"},
        {"role": "user", "content": "Very underwhelming experience."},
        {"role": "assistant", "content": "Negative"},
        {"role": "user", "content": "Great value for money![illegible]"},
        {"role": "assistant", "content": "Positive"},
        {"role": "user", "content": "Arrived damaged, very disappointed."},
        {"role": "assistant", "content": "Negative"},
        {"role": "user", "content": review}
    ]

    response = client.chat.completions.create(
        model="llama3-8b-8192",
        messages=messages,
        temperature=1,
        max_tokens=1024,
        top_p=1,
        stream=True,
        stop=None,
    )

    sentiment_result = ""
    for chunk in response:
        content = chunk.choices[0].delta.content or ""
        sentiment_result += re.sub(r"\s+", " ", content)  # Clean up the content

    return sentiment_result

# API endpoint to analyze the sentiment of uploaded reviews
@app.route('/analyze', methods=['POST'])
def analyze_reviews():
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded."}), 400

        file = request.files['file']
        reviews = read_reviews(file)

        # Initialize sentiment counters
        sentiment_counts = {"positive": 0, "negative": 0, "neutral": 0}

        for review in reviews:
            sentiment = get_sentiment(review)
            print(f"Review: '{review}' | Sentiment: {sentiment}")

            if "Positive" in sentiment:
                sentiment_counts["positive"] += 1
            elif "Negative" in sentiment:
                sentiment_counts["negative"] += 1
            elif "Neutral" in sentiment:
                sentiment_counts["neutral"] += 1

        return jsonify(sentiment_counts)

    except ValueError as ve:
        print(f"Value Error: {ve}")
        return jsonify({"error": str(ve)}), 400
    except Exception as e:
        print(f"Unexpected Error: {e}")
        traceback.print_exc()
        return jsonify({"error": "An error occurred during processing."}), 500

# Error handler for 404 errors
@app.errorhandler(404)
def handle_not_found(error):
    return jsonify({"error": "Endpoint not found."}), 404

# Run the Flask application
if __name__ == '__main__':
    app.run(debug=True)
```

Output:



```
{
    "negative": 23,
    "neutral": 7,
    "positive": 21
}
```

Github Link: https://github.com/AravindReddy16/Sentiment-Analysis-API-with-LLM-Integration